UNIVERSITA' DEGLI STUDI DI PISA

FACOLTA' DI SCIENZE MATEMATICHE FISICHE E NATURALI

CORSO DI LAUREA SPECIALISTICA IN TECNOLOGIE INFORMATICHE

TESI DI LAUREA

Progettazione e realizzazione di un protocollo di sicurezza per l'eXtensibile Metadata Hash Table

CANDIDATO

Giuseppe Di Francesco

RELATORE

Prof. Maurizio Bonuccelli

Dott. Claudio Cicconetti

CONTRORELATORE

Prof. Roberto Grossi

Anno Accademico 2010/2011

A mio Padre e mia Madre

Abstract

La diffusione delle tecnologie di informazione e comunicazione (ICT) consentirà nel futuro il miglioramento della qualità della vita per gli individui, specialmente per le persone anziane o con disabilità, consentendo loro una maggiore efficienza e produttività grazie a una aumentata autonomia, autostima e mobilità. Tuttavia, la creazione di piattaforme di Ambient Assisted Living (AAL) [26] [1] richiede funzionalità non presenti nella Internet al giorno d'oggi, come ad esempio la localizzazione di contenuti provenienti da oggetti del mondo reale o in mobilità. Tali funzionalità possono essere introdotte solo grazie a soluzioni *overlay network* [2], che però risultano comunque legate a paradigmi tradizionali nella sostanza e, quindi, poco flessibili o scalabili. Vi è quindi la necessità di una nuova struttura per l'Internet del futuro, un nuovo paradigma focalizzato sull'informazione. Nel nostro futuro ci saranno telefonini in grado di monitorare i parametri vitali del portatore e di trasmetterli in tempo reale al centro ospedaliero di riferimento, sistemi che rilevano i livelli di traffico all'interno di una galleria avvisando i soccorsi in caso di allarme, frigoriferi che controllano e segnalano la scadenza degli alimenti e automobili che dialogano con i semafori. Questi sono solo alcuni esempi delle infinite applicazioni di Internet 3.0 [17], che metterà in rete tra

[1] Programma comunitario congiunto istituito attraverso lŠarticolo 169 del Trattato dellŠUnione Europea come unione di diversi programmi di ricerca nazionali, al fine di supportare progetti per lo sviluppo di soluzioni innovative basate sulle tecnologie dellŠinformazione e della comunicazione volte a migliorare le condizioni e la qualità di vita delle persone anziane.

[2] rete logica definita tra i nodi, creando un rete sovrapposta (overlay network) sfruttando i collegamenti logici per lo scambio dei contenuti distribuiti tra gli stessi

loro non solo i computer, ma oggetti diversi e di uso quotidiano, dotati di una propria intelligenza e capaci di comunicare tra loro attraverso piccoli sensori che trasmettono e ricevono informazioni. Il risultato sarà un universo fisico interconnesso, che è stato ribattezzato *The Internet of Things* [18] [3].

Nel 2020, secondo l'Economist [4], passeremo dai 2 miliardi di persone connesse oggi a 50 miliardi di devices connessi, che scambieranno informazioni con la Rete [18].

In questo contesto un aspetto di particolare rilevanza è quello della sicurezza, intesa come confidenzialità, integrità e autenticazione. Per questo motivo è stato implementato un protocollo di autenticazione capace di garantire l'autenticazione dei nodi che faranno parte (join) della rete XMHT realizzata presso il laboratorio di ricerca e sviluppo della Intecs S.p.a sezione Telecomunicazioni e inoltre una libreria chiamata LEMS che si occupa di garantire l'integrità e la confidenzialità dei dati scambiati. Con l'auspicio che il futuro ormai vicino si preoccupi anche della privacy e della sicurezza dei dati confidenziali. Si pensi ad esempio ai dati clinici personali e ai possibili effetti di una diffusione impropria di essi, nonchè alle possibili implicazioni sociali e morali collegate [19].

[3] vedi 2.1

[4] The Economist è un settimanale con articoli di informazione da tutto il mondo edito a Londra da The Economist Newspaper Limited. La prima pubblicazione risale al 1843

Indice

Elenco delle figure

Capitolo 1

Introduzione

L'ecosistema di Internet esistente è il risultato di decenni di evoluzione. Esso è riuscito ad andare ben oltre le aspirazioni originali. L'evoluzione, però, ha evidenziato un certo grado di inadeguatezze ben documentate. Vi è quindi la necessità di una nuova struttura per l'Internet del futuro, un nuovo paradigma chiamato l'Internet delle Cose (IoT) [1].

L'IoT è la terza generazione di Internet, identificata in NetInf [2]che cercherà di focalizzare l'attenzione su quello che interessa di più agli esseri umani, ovvero le informazioni.

Consiste nell'interconnessione degli oggetti stessi che possono acquisire un ruolo attivo grazie al collegamento alla Rete. L'obiettivo è far sì che il mondo elettronico tracci una mappa di quello reale, dando una identità elettronica alle cose e ai luoghi dell'ambiente fisico.

Nel 2020, secondo l'Economist, passeremo dai 2 miliardi di persone connesse oggi a 50 miliardi di devices connessi, che scambieranno informazioni con la Rete.[19]

Questo è molto interessante non solo per il numero elevato di connessioni previste in così poco tempo,(solo alcuni anni fa era impensabile una evoluzione del genere nel giro di 10 anni) ma bensì alla tipologia di connessioni future, non più utenti ma oggetti, devices e non solo.

[1]Internet of Things vedi paragrafo 2.1

[2]Networking of Information vedi paragrafo 2.2

L'Internet of Things porta molti vantaggi, riguardo l'efficenza, la flessibilità, la semplicità ma soprattutto può includere degli aspetti di sicurezza intrinseci al nuovo paradigma. In quest'ottica diventa fondamentale pensare alla enorme quantità di dati scambiati tra i devices, per cui l'aspetto della sicurezza diventa cruciale nell'Internet delle cose. Lo studio di questa tesi si focalizzerà soprattutto su questo aspetto.
Con il nuovo paradigma information-centric, l'informazione come abbiamo visto acquisisce maggiore rilievo, quindi si ha la necessità di garantire la sicurezza di queste informazioni. Questo aspetto nell'Internet di oggi non è stato curato particolarmente, per cui la nuova visione di Internet potrebbe aiutare a colmare questo handicap della sicurezza molto tralasciata nei giorni nostri.

1.1 Descrizione del Problema

Nelle reti di oggi, la sicurezza riguarda principalmente la protezione delle comunicazioni tra gli host e le persone. L'integrità e l'autenticità dei dati effettivamente trasmessi sono solitamente stabiliti in maniera indiretta dalla fiducia di altre parti. Nella nostra visione di rete questo modello non è sufficiente. L'integrità e l'autenticità dei dati invece devono essere fornite dagli oggetti stessi, indipendentemente dall'host che fornisce l'oggetto.
L'integrità e l'autenticità possono essere direttamente legati ai nomi degli oggetti, con l'autocertificazione del nome: in questo modo c'è una relazione cifrata tra il nome e l'oggetto. Verificare l'autenticità è più difficile senza consultare una terza parte di fiducia come le Certification Autority. [3]
Più avanti vedremo i vari meccamismi proposti da alcuni studi europei, e analizzeremo nel dettaglio quelli utilizzati per raggiungere l'obiettivo propo-

[3] In crittografia, una Certificate Authority o Certification Authority (CA), letteralmente Autorità Certificativa, è un ente di terza parte (trusted third party), pubblico o privato, abilitato a rilasciare un certificato digitale tramite procedura di certificazione che segue standard internazionali

sto, cioè rendere sicure le informazioni nel nuovo modello dell'Internet del futuro.

1.2 Soluzione

La sicurezza della nuova Internet è vista in maniera diversa. Se da un lato oggi abbiamo la necessità di rendere sicure le infrastrutture, i nodi, i server e le comunicazioni, adesso dobbiamo concentrarci maggiormente sulle informazioni stesse, implementando dei meccanismi intrinseci al framework NetInf [4]. Chiaramente tutto questo è possibile fidandosi dei nodi che pubblicano le informazioni [5], (soltanto i nodi abilitati alla pubblicazione possono farlo) quindi verrà affrontato anche questo aspetto, cruciale per la sicurezza delle informazioni.
In questo senso la sicurezza può essere divisa in due parti:

- la prima, che si occupa dell'autenticazione dei nodi che comunicano: nel nostro caso permette ad un nuovo nodo di entrare a far parte della rete overlay XMHT[6].

- la seconda, invece, serve a garantire l'integrità e la confidenzialità delle comunicazioni, basandosi sui metadati di sicurezza presenti negli oggetti chiamati Information Object.

Nel capitolo 2 descriviamo nel dettaglio gli Information Object e i Data Object, in modo da avere una visione più completa del lavoro svolto. Dopo aver discusso dei benefici di una tale rete, si considera l'impatto di NetInf nella società del futuro. Infine, forniamo una prospettiva sul campo di applicazione della ricerca di NetInf per alcuni sviluppi futuri.

[4]Networking of Information, framework preso in considerazione per lo studio della tesi, vedi 2.2

[5]vedi paradigma pubblicazione/sottoscrizione par 2.2.1

[6]eXtensible Metadata Hash Table, è un overlay peer to peer basato su DHT, implementato dal gruppo di ricerca e sviluppo Intecs S.p.a con cui ho collaborato per il lavoro di tesi.

1.3 Struttura della tesi

La tesi è organizzata nel modo seguente:

Capitolo 1 in questo capitolo si spiegano le motivazioni per cui è stato svolto il lavoro di tesi, le problematiche riscontrate nell'affrontare il problema che si intende risolvere, e infine gli obiettivi di questa tesi riguardanti l'aspetto della sicurezza e la gestione degli Information Object attraverso i metadati.

Capitolo 2 descrive lo stato dell'arte dei concetti base del nuovo paradigma IoT e dell' overlay XMHT realizzato dal gruppo di ricerca e sviluppo della INTECS S.p.a sezione Telecomunicazioni, con cui ho collaborato per il lavoro di tesi. Inoltre, vengono descritti gli studi attuali sull'aspetto della sicurezza nell'ambito dell'IoT.

Capitolo 3 fornisce un'analisi dettagliata del protocollo di autenticazione realizzato e descrive la libreria LEMS [7], sviluppata per lo scambio sicuro di informazioni, attraverso la gestione dei metadati.

Capitolo 4 descrive i risultati ottenuti da questo lavoro di tesi, le relative conclusioni e i lavori futuri.

Appendice A Source Code di una piccola applicazione per testare il funzionamento della libreria Lems.

[7] Library Extensible Metadata Security

Capitolo 2

Stato dell'Arte e della Tecnica

In questo capitolo si descrive nel dettaglio lo stato dell'arte degli strumenti utilizzati per lo svolgimento della tesi.
Inizialmente si ha una breve panoramica esplicativa sull'Internet delle cose, raccontando la sua genesi, lo stato attuale e futuro. Nel secondo paragrafo si spiegano i concetti base sul paradigma NetInf[1], il terzo è dedicato allo stato dell'overlay xmht, il quarto e il quinto rispettivamente descrivono gli obiettivi da raggiungere e i loro studi attuali.

2.1 Internet of Things

Nel nostro futuro ci saranno telefonini in grado di monitorare i parametri vitali del portatore e di trasmetterli in tempo reale al centro ospedaliero di riferimento, sistemi che rilevano i livelli di traffico all'interno di una galleria avvisando i soccorsi in caso di allarme, frigoriferi che controllano e segnalano la scadenza degli alimenti e automobili che dialogano con i semafori. Questi sono solo alcuni esempi delle infinite applicazioni di Internet 3.0, che metterà in rete tra loro non solo i computer, ma oggetti diversi e di uso quotidiano, dotati di una propria intelligenza e capaci di comunicare tra loro attraverso

[1]Networking of Information è la visione del paradigma information-centric emerso dal progetto 4WARD, da cui nasce l'implementazione della Libreria LEMS (Library Extensible Metadata Security)

piccoli sensori che trasmettono e ricevono informazioni. Il risultato sarà un universo fisico interconnesso, che è stato ribattezzato The Internet of Things [17].

2.1.1 Genesi

I primi studi sull'IoT risalgono al 1999, condotti da parte di un gruppo di ricercatori dell'Auto-ID Center presso il *Massachusetts Institute of Technology MIT*, che si occuparono principalmente di sistemi di identificazione a radiofrequenza con l'obiettivo di dotare gli oggetti di intelligenza e consentire il monitoraggio durante il loro ciclo di vita.
Da allora, lo sviluppo di Internet è stato travolgente: il numero di dispositivi connessi è aumentato e i 4 miliardi di indirizzi generabili con IPv4 potrebbero non bastare.
Per ovviare a questo problema l'ICANN [2] ha recentemente disposto il passaggio dal protocollo IPv4 all'IPv6: il nuovo standard consente di moltiplicare le possibilità di accesso alla rete portando il numero di indirizzi disponibili a 2^{128}, e quindi permette l'accesso non solo ai computer e simili, ma anche ad oggetti intelligenti di varia natura. Presto più di un triliardo di device saranno potenzialmente connessi fra loro.
La sfida dell'Internet degli oggetti, o delle cose, è quella di abilitare a parlare con un linguaggio comune le reti di sensori e tutti quei dispositivi dotati di vari livelli di intelligenza. Questo consentirebbe di costruire applicazioni e servizi complessi a partire dall'aggregazione dei dati forniti dai componenti della rete.
Occorre sottolineare che non esiste un unico mercato per le soluzioni che si basano sull'IoT, ma una pluralità di mercati per applicazioni verticali dove però, fino ad oggi, i player principali hanno fornito per lo più soluzioni chiuse e proprietarie finalizzate a risolvere specifici problemi.

[2] Internet Corporation for Assigned Names and Numbers

Una soluzione studiata dal progetto The FP7 4WARD Project è l'architettura NetInf, di cui approfondiremo nei paragrafi seguenti.

2.2 NetInf

Questo paragrafo descrive gli studi riguardo al progetto 4WARD finanziato dall'UE [21] di cui una piccola parte è questo lavoro di tesi.
Mentre il web è iniziato come una raccolta di documenti interconnessi, Internet oggi è piena di una vasta collezione di oggetti, di dati incoerenti fra di loro, provenienti da reti P2P, wiki, community, siti di video, e altre pagine web. Molti di questi dati si riferiscono allo stesso argomento, ma questo rapporto non è rappresentato in alcun modo su Internet. Allo stesso modo, le varie codifiche diverse (ad esempio MP3, WMA) degli elementi fanno sì che copie esatte delle stesse informazioni non sono rappresentati nella Internet di oggi.

L'incapacità di Internet ad oggi nel rappresentare la relazione tra questi elementi strettamente connessi, ha almeno due principali svantaggi. In primo luogo, il recupero delle informazioni è inutilmente complicato. Mentre gli utenti esperti sono abituati a utilizzare i motori di ricerca, wiki, le comunità, le reti P2P, e così via, per trovare informazioni su Internet, questo rappresenta un ostacolo per gli utenti meno esperti. In secondo luogo, la diffusione delle informazioni è inefficiente. Per esempio, le copie esistenti di oggetti non possono essere utilizzate efficientemente per sostenere un processo di diffusione dei dati derivanti dal loro rapporto, attualmente sconosciuto. Sono state proposte diverse soluzioni, come l' overlay Bittorrent, cercando di ridurre tali svantaggi, derivanti dal fatto che Internet è stato originariamente progettato per comunicazioni point-to-point e non per la diffusione dei dati. Oggi, tuttavia, la stragrande maggioranza degli utenti utilizza internet per diffondere i dati, e non per lo scambio di informazioni o per comunicazioni point-to-point come sottolineato da Van Jacobson [8]. In altre parole, oggi

Internet viene utilizzata solo come un'eccezione alla regola. Inoltre, Internet di oggi non è molto adatta per la diffusione di informazioni attendibili e affidabili. Nasce quindi la necessità di una nuova architettura di rete basata sull'informazione.

In NetInf, il contenuto, o oggetti informativi, sono i principali attori della rete, indipendentemente dai dispositivi su cui sono memorizzati. Si crede che questo approccio può soddisfare appieno le esigenze delle applicazioni che dominano l'uso delle reti attuali, ossia la distribuzione delle informazioni. Inoltre, fornisce un servizio migliore per la connettività e la protezione dal traffico dannoso e indesiderato. Al contrario, le reti attuali sono basate sull' interconnessione di diversi dispositivi, computer, dispositivi mobili, server e router. In NetInf, l'identità delle informazioni o oggetti è indipendente dal dispositivo su cui sono memorizzati.
NetInf introduce i concetti di oggetti informativi [3] e dei dati [4], da qui in poi chiamati rispettivamente IO e DO. I data object corrispondono alla rappresentazione digitale dell'oggetto, esempio un file di testo, una specifica codifica di una canzone o un video etc. Gli oggetti informativi invece vengono utilizzati per identificare i corrispondenti DO, indipendentemente dalla loro posizione e rappresentazione digitale specifica.

2.2.1 Struttura NetInf

NetInf tratta le informazioni, come abbiamo accennato sotto forma dei cosiddetti Information Object. Gli IO possono essere considerati i principali tipi di oggetto, in quanto sono i primi ad essere riferiti direttamente da NetInf. Un esempio concreto di un IO sono le pagine web e le mail. Tuttavia, per una vera rivoluzione della rete, occorre includere i servizi di streaming e in tempo reale, telefonia sia audio che video e infine la rappresentazione

[3] Information Object
[4] Data Object

virtuale degli oggetti fisici. L'obiettivo è di rendere tutte le informazioni facilmente disponibili dall'utente.

I Data Object invece rappresentano il secondo livello di informazione, contenente il vero contenuto del dato e vengono riferiti dagli IO. Questi di solito sono file, stream o altre rappresentazioni di dati in un formato specifico, come un file mp3 con una certa codifica. Tali oggetti possono essere ulteriormente suddivisi in blocchi, più piccoli, detti chunk, per supportare la funzionalità di download simultanei. In molti casi, tuttavia, un utente non è realmente interessato ai dati specifici del DO (tipo codifica di un brano mp3), ma alle informazioni della sua rappresentazione (brano,nona sinfonia di Beethoven). Queste informazioni sono ricavate dalla semantica espressa dagli IO del livello superiore. Un IO può, ad esempio, riferirsi ad una certa canzone senza specificare la codifica o l'orchestra che la esegue. Gli IO consentono agli utenti di trovare contenuti indipendentemente dalla sua specifica rappresentazione e indipendentemente da alcune determinate caratteristiche che potrebbero non interessare all'utente. Inoltre gli IO possono essere composti da altri IO o riferirsi direttamente a uno o più DO che contengono il contenuto reale.

I *Metadati* ci permettono di esprimere il significato semantico degli IO, descrivono il contenuto e/o la sua relazione con altri oggetti. La ricerca esistente in questo campo fornisce un ottimo punto di partenza per l'integrazione di queste caratteristiche nella rete, con particolare riferimento alla descrizione dei linguaggi come Resource Description Framework [22] o la capacità di stabilire automaticamente le relazioni tra gli IO. Vedendo in questo modo gli IO, è facilmente possibile reperirli in streaming da tutto il mondo NetInf. Ad esempio, un episodio di una serie televisiva potrebbe essere rappresentato da un IO. Con un grande margine di libertà offerto da NetInf, si potrebbe associare un riferimento dell'episodio ancora inedito (non ancora

pubblicato sotto forma di IO) e quando l'episodio viene pubblicato, modificare soltanto l'associazione tra il riferimento e l'episodio effettivo. Quando l'associazione cambia, gli utenti possono essere avvisati, in modo esplicito mediante l'invio di una notifica o implicitamente inviando il contenuto appena associato. I DO che contengono il video possono essere i file che vengono scaricati o streaming multimediali. Allo stesso modo, anche una conversazione telefonica potrebbe essere rappresentata da un IO,(con voce e streaming video). Un IO potrebbe riferirsi anche a qualcosa non ancora associato, ad esempio, nel caso di uno streaming live non ha ancora iniziato.

Versioni

Un ulteriore sfida è la gestione degli IO dinamici. Possono esistere diverse versioni dello stesso IO, basti pensare alla pagina del New York Times o di qualsiasi altro giornale che cambia continuamente di giorno in giorno. La soluzione più semplice è quello di associare ad ogni IO un diverso DO. In questo modo quando necessario, l'IO può cambiare l'associazione alla nuova DO. Il vecchio può essere associato all'IO del giorno precedente. Le versioni possono essere un attributo esplicito di un IO. Per gli oggetti che hanno nomi autocertificati (hash del file), non è possibile avere diverse versioni dello stesso oggetto, in quanto ogni nuova versione, per definizione, otterrebbe un nuovo nome. L'eliminazione e la revoca dell'oggetto NetInf sono problemi difficili. Consentire l'eliminazione di tutte le copie disponibili dell'oggetto, richiederebbe un registro centrale che ne tiene traccia. Inoltre, a fronte delle possibili disconnessioni, non è possibile garantire la coerenza delle copie. Per affrontare il problema della consistenza bisogna pensare ad un'architettura dove gli oggetti possono essere invalidati da soli. Per implementare questo tipo di invalidazione, devono essere presenti nei metadati degli attributi che permettono la verifica della validazione dell'oggetto. Questi attributi possono essere chiavi crittografiche, certificazioni e firme digitali.

Paradigma di pubblicazione/sottoscrizione

Il paradigma di pubblicazione/sottoscrizione [9] è molto attraente come modello di interfaccia o di interazione per una rete di informazioni. In questo paradigma, i destinatari indicano il loro interesse a ricevere eventi particolari mediante la sottoscrizione di tali eventi. I Mittenti indipendentemente pubblicano gli eventi, che si traduce nei destinatari con sottoscrizioni, ottenendo una notifica di corrispondenza. Il paradigma fornisce un disaccoppiamento tra le parti comunicanti, il mittente e il destinatario, sia nel tempo che nello spazio. La parte fondamentale di un sistema di pubblicazione/sottoscrizione è il servizio di notifica degli eventi, che fornisce la gestione e l'archiviazione degli iscritti per recapitare le notifiche dell'evento. Il meccanismo utilizzato per associare gli eventi con le sottoscrizioni è cruciale per la funzionalità, nonché per le prestazioni e la scalabilità.

2.2.2 Componenti di NetInf

Una funzionalità chiave di NetInf è il recupero degli oggetti, basato sui loro identificatori univoci. Questo processo include in genere due fasi principali, descritti nelle prossime due sottosezioni. In primo luogo, la risoluzione dei nomi che individua l'oggetto nella rete, il routing dell'oggetto, e infine il recupero del rispettivo DO all'utente finale.

2.2.3 Architettura di NetInf

In NetInf l'indirizzo IP viene diviso in due concetti logici, chiamati Identificatore (ID) e Locatore (L) rispettivamente. In questo modo la risorsa può essere identificata in maniera univoca, indipendentemente dalla sua posizione fisica, portando molti benefici soprattutto sulla mobilità e sulla gestione degli oggetti applicata all'IoT.
Si assume che ogni risorsa viene identificata tramite il suo ID univoco nella rete, e la funzione di mapping della risorsa è conosciuta da tutti gli iscritti. Quando una nuova risorsa entra nella rete, viene pubblicato il suo Identifi-

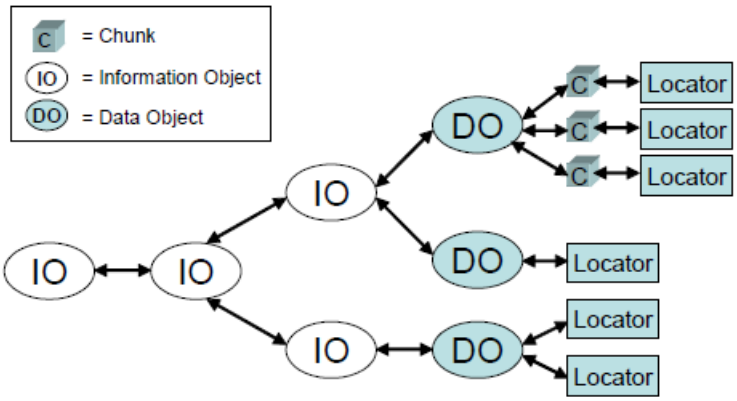


Fig. 2.1: *Relazioni tra i componenti di NetInf*

catore nella rete, e viene associato al suo Locatore:

$$\{ID_1\} \longrightarrow \{Locatore_1\}$$

Se il locatore cambia, perchè per esempio la risorsa è mobile, il suo locatore viene aggiornato alla rete ma il suo Identificatore rimane lo stesso. Quindi quando un cliente richiede la risorsa, verrà identificato con l'ultimo locatore aggiornato. Questo processo viene descritto nella figura 2.2. Se lo stesso servizio è fornito da più Fonti nella rete, un Identificatore si può riferire a più locatori. Un esempio può essere il caso in cui un turista richiede delle informazioni su un determinato edificio storico il quale è mappato con più Identificatori. Nel caso ci siano più informazioni sull'edificio proveniente da più Sorgenti, allora la ricerca ritornerà una lista di locatori {locatore1,locatore2 ..}

Per un implementazione della rete sopra descritta si è scelta un'architettura basato sull'overlay P2P [11]. Molti studi hanno dimostrato che l'architettura P2P è adatta alle reti mobili e in linea ai nostri concetti, in quanto permette scalabilità, affidabilità e inoltre immune agli attacchi di tipo Denial of Ser-

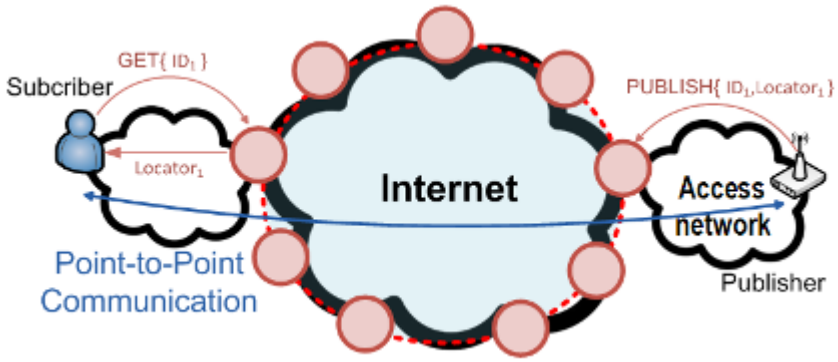


Fig. 2.2: *esempio di risoluzione del locatore in un servizio a paradigma publish/subscriber*

vice (DoS) [5] [3]. Questo tipo di attacco è comune nell'Internet di oggi ma per la loro natura distribuita non può essere eseguito sulle reti P2P.

Il concetto di base del P2P è il Distributed Hash Table (DHT). Una tabella hash è formata da una coppia (key,value), dove ad ogni chiave corrisponde un valore. Il concetto di fondo delle tabelle Hash vale anche per le DHT, a differenza che le chiavi non vengono memorizzate nello stesso nodo, ma sono distribuite uniformemente in più nodi, chiamati peers.

Nel nostro caso la chiave è l'hash del nome, quindi l'dentificatore, mentre il valore è il suo locatore. I peers sono tutti i nodi che possono accedere alla rete pubblicando o sottoscrivendo le informazioni. Tuttavia i dispositivi mobili potrebbero essere soggetti ad alcuni limiti tecnologici, tipo capacità di memoria, basse prestazioni dovute proprio alla natura mobile. Quindi, assumiamo che i nodi abilitati alla pubblicazione di informazioni siano soltanto i nodi capaci di garantire questa stabilità, come gli access point, base station, router oppure dispositivi wireless a patto che siano statici.

Le fasi di pubblicazione e recupero dei dati è illustrata nella figura 2.3 seguente. Dal lato sinistro della figura, vediamo un service provider che si connette ad un peer della rete P2P, chiamato *nodo ancora*, per pubblicizza-

[5] è un tipo di attacco molto frequente nei servizi web, si tratta di saturare il server inondandolo di richieste fino a renderlo non più in grado di erogare il servizio

re(indicizzare) un locatore associato al nome dato. Il *nodo ancora* ottiene la chiave applicando la funzione hash del nome, questa chiave viene quindi usata attraverso un processo iterativo per trovare il nodo chiamato *home node*. *Home node* è il peer reponsabile dell'attuale memorizzazione del locatore. Poiché tutti i peer utilizzano lo stesso processo di ricerca, ogni richiesta della stessa chiave iniziata da un nodo mittente, *forwarder node*, finalmente raggiungerà l' home node, il quale può rispondere direttamente ad esso, come raffigurato nella parte destra della figura. Questo meccanismo è supportato in quanto ogni peer tiene traccia delle associazioni chiave-locatori in una tabella. Un ulteriore ottimizzazione durante il processo di pubblicazione, potrebbe essere il salvataggio da parte dei nodi intermedi compreso il nodo ancora dei riferimenti delle ricerce in alcune cache, ma questo comporterebbe altri meccaniscimi aggiuntivi per la gestione della coerenza. Una dimostrazione teorica e pratica sui DHT [12], asscicura che i locatori vengono distribuiti nella rete uniformemente, e inoltre il numero di hop tra i nodi ancora/forwarder e il nodo home è al massimo $\log_2 B(N)$ [13], dove B è il numero di bit per la rappresentazione della chiave e N il numero di peer. Questa proprietà garantisce la scalabilità dei sistemi P2P, in quanto il numero di hop che impatta sul traffico della rete e sulla latenza delle operazioni di retrieval/publishing, aumentano logaritmicamente rispetto al numero di nodi. Inoltre possono essere implementate vari livelli di DHT, per raggiungere una maggiore scalabilità geografica, utile per servizi come un sistema intelligente di trasporto dove ad esempio i veicoli accedono ai servizi (informazioni su trasporti pubblici, POI, informazioni sulla mappa locale, sul posto auto libero più vicino). In questo senso è necessario richiedere le informazioni al nodo più vicino possibile. Questo tipo di meccanismo può essere implementato con vari livelli di DHT gestiti gerarchicamente.

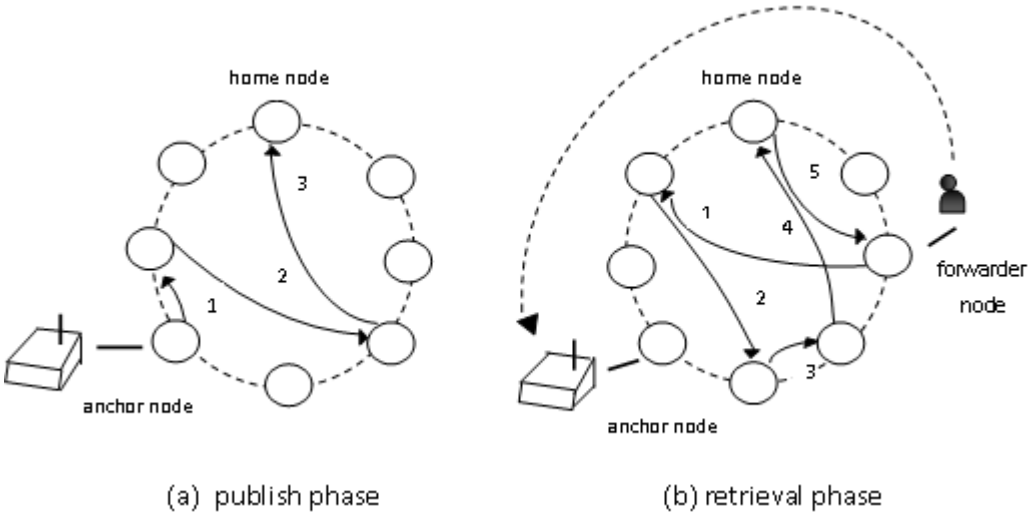


Fig. 2.3: *implementazione basata su DHT*

2.2.4 Naming Framework

NetInf è stato progettato in modo da essere sicuro per qualsiasi tipo di contentuto, incluse pagine Web, file multimediali, e flussi di dati in tempo reale. Può essere usato per rappresentare in modo sicuro servizi, persone e oggetti del mondo reale. Il termine generico usato per una parte del contenuto o informazione è l'Information Object (IO).
Nel Framework NetInf, un IO ha un nome unico a livello globale, il quale sarà riferito dal suo Identificatore. L' Information Object insieme ai suoi dati e ai suoi metadati rispettivamente può essere definito come:

$$IO = (ID, Dato, Metadato)$$

Il Dato contiene le principali informazioni di contenuto dell' Information-Object. I Metadati contengono alcuni attributi associati all'Information Object che non sono presenti nel Dato e nemmeno nell'Identificatore. Il Metadato contiene informazioni necessarie per eseguire funzioni di sicurezza del framework NetInf, come chiavi pubbliche, contenuto hash, e certificati. Esso

inoltre contiene una firma di dati che autentica il contenuto. Il Metadato può essere parte integrante di un IO oppure nel nostro caso è separato, e viene trattato e salvato indipendentemente. Naturalmente il Metadato di un generico IO può essere formato da più metadati, quindi possono esistere più metadati associati allo stesso IO. Lo stesso IO esiste tipicamente in molte copie in diverse posizioni, ma con la separazione dell'Identificatore e del Locatore, queste copie di IO hanno gli stessi nomi in NetInf. In particolare possono corrispondere a diverse versioni dello stesso IO. Tutte le IO con lo stesso nome sono considerati come reciprocamente equivalenti e sono, quindi, considerate in una classe di equivalenza, che viene cosi' definita e rappresentata dal nome comune. In NetInf gli IO sono manipolati (es. sono generati, modificati, registrati, e recuperati), da entità fisiche come i nodi, sia che siano clienti, persone, o compagnie. Dobbiamo quindi distinguere diversi ruoli di tali entità. Le entità fisiche in grado di generare, creare o modificare gli oggetti sono i proprietari chiamati *owners*. Gli owners sono autori/creatori di contenuti. Naturalmente IO differenti possono avere diversi owners, esempio differenti versioni sotto lo stesso nome. I proprietari inoltre hanno i loro nomi, con la stessa struttura dei nomi degli IO nel nostro framework. Oltre ai proprietari, esistono gli editori detti *publishers* che sono autorizzati dai proprietari a registrare, pubblicare e distribuire il contenuto. Tuttavia, gli editori in genere non sono autorizzati a modificare o firmare gli IO.

2.3 Stato dell'overlay

L'overlay utilizzato dalla Intecs S.p.a per l'implementazione di NetInf è **eXtensible Metadata Hash Table (XMHT)** realizzato dal gruppo ricerca e sviluppo della Intecs S.p.a sezione Telecomunicazioni, presso cui si è svolto il lavoro di tesi.

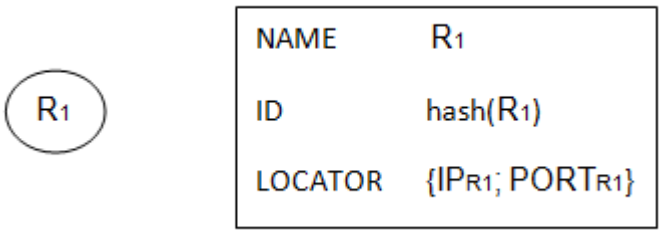
L'overlay XMHT va ad inserirsi all'interno di un'applicazione di [meta]data

retrieval, in particolare nella fase di ricerca delle informazioni che portano a poter richiedere la risorsa desiderata: tali informazioni sono dette **metadati**. Si tratta di informazioni in formato XML che, a parte un set minimo di parametri necessari, possono contenere informazioni application-dependent.

Sia i nodi che intendono esportare una risorsa, sia quelli che intendono richiederne una, non devono essere necessariamente connessi alla rete XMHT: non devono cioè essere peer dell'overlay. L'unico requisito per entrambe le categorie è quello di essere a conoscenza di un endpoint dell'overlay al quale riferirsi.

Al fine di individuare una risorsa si utilizza il concetto di *Identificatore*: esso identifica univocamente una risorsa nell'overlay; il *Locatore* invece ne dà la posizione nella rete.

Così come le risorse, anche i nodi della rete XMHT hanno associato un



identificatore selezionato dal medesimo spazio degli identificatori delle risorse

2.3.1 Protocollo di data Retrieval

Per meglio comprendere il funzionamento dell'architettura, si consideri il seguente scenario in cui un 'applicazione sul nodo C voglia richiedere una risorsa R_1 locata sul nodo R, non essendo a conoscenza del suo locatore

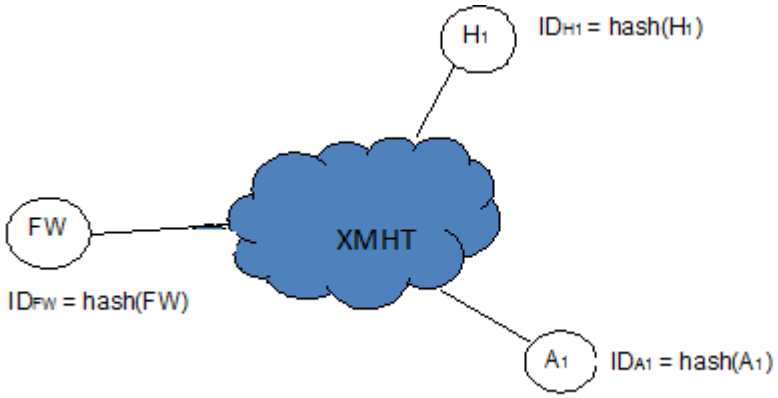


Fig. 2.4: *esempio rete XMHT*

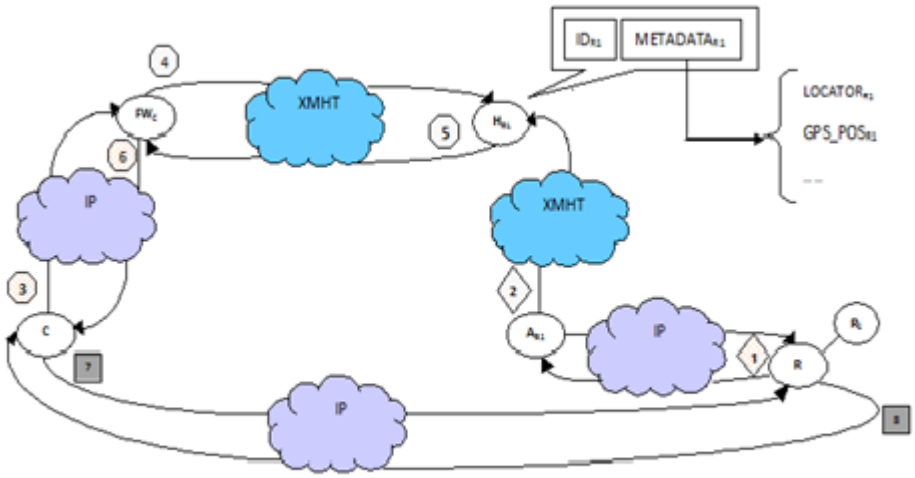


Fig. 2.5: *scenario di richiesta di una risorsa R1 locata sul nodo R*

Innanzitutto, il nodo R pubblica la risorsa R_1 inviando un messaggio(1) di PUBLISH_REQUEST ad un *endpoint* dell'overlay: tale nodo sarà il *nodo ancora* della risorsa (nell'esempio A_{R1}) all'interno della rete XMHT per la risorsa pubblicata. L'instradamento del messaggio fra questi due nodi av-

viene mediante IP standard. Il messaggio di PUBLISH_REQUEST contiene sia l'ID della risorsa che i metadati ad essa associati. Nell'esempio considerato, conterranno sia il locatore (facente parte del set minimo di parametri che ciascun metadato deve contenere) sia la posizione GPS della risorsa (opzionale).

Facendo parte dell'overlay XMHT, il nodo ancora pubblica nella rete la risorsa inviando (2) un messaggio di PUBLISH: ad ogni hop, tale messaggio sarà inviato verso il nodo avente ID numericamente più vicino all'ID della risorsa pubblicata. Il messaggio arriverà infine su un nodo dell'overlay che non sarà in grado di trovare nella sua routing table un nodo avente identificatore numericamente più vicino: tale nodo sarà il *nodo home* della risorsa (nell'esempio H_{R1}) ed inserirà all'interno della sua resource table una entry contenente l'ID della risorsa ed il suo metadato.

Quando un nodo C avrà necessità di richiedere la risorsa R_1, invierà un messaggio (3) di GET ad un endpoint dell'overlay: tale nodo sarà il *nodo forward* del nodo richiedente(nell'esempio FW_c) all'interno della rete XMHT. L'instradamento del messaggio fra questi ue nodi avviene mediante IP standard.

Facendo parte dell'overlay XMHT, il nodo forward richiede nella rete la risorsa inviando (4) un messaggio di META_REQUEST: ad ogni hop tale messaggio sarà inviato verso il nodo avente ID numericamente più vicino all'ID della risorsa richiesta, fino a raggiungere il nodo H_{R1}.

Compilando il messaggio META_RESPONSE (5), che verrà inoltrato verso il nodo forward in base al relativo identificatore, il nodo home inserirà il metadato associato come payload.
A tal punto il nodo forward invierà sulla rete IP standard un messaggio (6)

di REPLY al nodo C contenente i metadati per la risorsa richiesta, nodo che potrà mediante i messaggi (7) ed (8) ottenere la risorsa dal nodo R mediante una richiesta IP standard, avendo a questo punto a disposizione il locatore.

I messaggi (1), (3) e (6), sono in generale inviati fra endpoint di una rete IP standard al fine di garantire:

1. interoperabilità con applicazioni vendor specifici. A tal proposito è stata fornita una interfaccia Java che fornisce un API alle applicazioni utente: tale interfaccia ha l'obiettivo di rendere trasparente l'invio dei messaggi da e verso la rete XMHT mediante delle chiamate di funzione

2. interoperabilità della rete XMHT con la rete IP tradizionale, garantendo un'estensibilità futura dell'overlay

2.4 Sicurezza dell'overlay

L'overlay XMHT è capace di allocare e recuperare le risorse in maniera efficiente proprio per la struttura DHT della rete, ma in nessun modo tiene conto dell'aspetto della sicurezza. Obiettivo di questa tesi è quindi rendere la pubblicazione e il retrieval dei dati in maniera sicura e affidabile. Di seguito vedremo i vantaggi e gli svantaggi di alcune proposte per la soluzione di questo problema, evidenziando i motivi per cui si è scelto di seguire la proposta NetInf.

DONA (Data-Oriented Network Architecture) associa la chiave pubblica ad un proprietario, e non all'IO, di conseguenza, se un proprietario cambia, cambia anche l'Identificatore del IO [3]. Questo significa che la proprietà della persistenza del nome non è soddisfatta in DONA. Al fine di raggiungere la persistenza del nome, la chiave pubblica deve essere associata all' IO stesso, in particolare, può corrispondere al primo proprietario e non all'owner attuale. Un'altra caratteristica presente in NetInf ma non in DONA

è che l'autenticazione e/o l'identificazione dell'owner è separata dall'autenticazione contenuta nella chiave pubblica usata per l'oggetto. Per questo può essere differente dall'Identificatore di un IO. In particolare, questo implica che l'autenticazione/identificazione dell'owner è possibile anche se il proprietario cambia, pur mantenendo lo stesso Identificatore dell'IO. Tra tutti i creatori di un IO, soltanto alcuni di loro possono essere autenticati/identificati, mentre gli altri possono rimanere anonimi.

Gli stessi vantaggi sono validi anche rispetto al SDSI (Simple Distributed Security Infrastructure) [3]. Un ulteriore vantaggio di NetInf rispetto a DONA è la flessibilità offerta dai tipi di Identificatori variabili.

CCN (Content Centric Networking) usa nomi gerarchici per gli oggetti che vengono divisi in *data packets*. In questo modo, consente il routing dei dati per nome senza bisogno di registrarli. La strutturra dei nomi è essenzialmente come l'URL, dove la parte principale del nome è analoga al nome del dominio globale nell' URL e tipicamente corrisponde a strutture organizzative e il resto del nome è analogo al path locale. L'autenticazione del nome e dei contenuti viene eseguita da firme digitali da parte di soggetti che possono o non possono essere associati ai nomi dei dati. Questa autenticazione è condizionata dalla fiducia di queste entità e dalle loro chiavi pubbliche. Se il proprietario del dato cambia, allora cambia anche il nome del dato e sua volta anche la fiducia nella firma del soggetto, in quanto la firma dell'entità è associata al nome del dato. Tuttavia, le firme non sono limitate ad essere associate soltanto ai nomi dei dati. Questo significa che a differeza di NetInf, la fiducia necessaria in CCN, non è legata ai nomi dei dati e, quindi difficile da gestire. In NetInf questo non succede in quanto le chiavi pubbliche dei proprietari sono presenti nei nomi dei metadati. Inoltre, poichè le chiavi pubbliche utilizzate per la firma sono collocati al di fuori della struttura del nome, gli utenti non possono specificare nel loro interesse le chiavi pubbli-

che in anticipo, questo evita quindi di essere sopraffatti da falsi pacchetti di dati(fake) con il nome giusto. Inoltre, dato che non vi è alcuna autenticazione del nome nella registrazione, il sistema CCN può essere floddato da possibili pacchetti fake causando attacchi di tipo DoS (Denial of Service). Di conseguenza la struttura del nome in CCN non possiede funzionalità di sicurezza intrinseche.

Per i motivi sopra si è scelto di adottare il framework NetInf, e questa tesi segue le sue specifiche. Il framework NetInf, costruisce le fondamenta per un modello information-centric sicuro, che integra la sicurezza fino in profondità nell'architettura. In questo modello, la fiducia è basata sull'informazione stessa.
Con la divisione dell'Identificatore e del Locatore, ogni oggetto è rappresentato da un nome unico, avente proprietà crittografiche. Con l'aggiunta dei metadati, il nome può essere usato per verificare l'integrità del dato, e altre proprietà non solo crittografiche, riguardanti la sicurezza. Questo nuovo approccio offre anche altri vantaggi rispetto al modello di sicurezza dell'internet di oggi: ad esempio riduce al minimo la necessità di una fiducia nelle infrastrutture, negli host, nei canali di trasmissione e nella risoluzione dei servizi come ad esempio il DNS.
Alcuni elementi del framework, come il nome dei tipi e le versioni sono lasciate come facoltative al fine di garantire la flessibilità e l'estensibilità, che sono importanti per il futuro delle reti con il nuovo pardigma information-centric, specialmente per le applicazioni che sono più versatili e dinamiche.

2.5 Definizione obiettivi

Il framework NetInf essenzialmente definisce delle proprietà da essere soddisfatte, l'obiettivo della libreria LEMS è soddisfare alcune di queste proprietà. Possono essere classificate come principali o proprietà aggiuntive.

Le proprietà principali sono :

- auto - certificazione
- persistenza del nome
- autenticazione del proprietario

mentre le proprietà aggiuntive sono:

- autenticazione del nome
- certificazione del dato

Si noti che la maggior parte di queste proprietà sono legate alla sicurezza.

L' autocertificazione viene chiamata s-certData in un IO e garantisce l'integrità del dato e/o del metadato, a condizione che l'Identificatore dell'oggetto è corretto, cioè autentico. In particolare, l'Identificatore può essere conosciuto come autentico in anticipo, da una terza parte di fiducia. Alternativamente, l'autenticità del nome può essere fornita dal nome certificato. In tutti e due i casi questa operazione deve essere effettuata dall'applicazione e non può in alcun modo essere effettuata dalla libreria per ovvi motivi. Più precisamente questa proprietà dovrebbe individuare i tentativi di cambiamento di s-certData non autorizzati.

La seconda proprietà da soddisfare è la persistenza del nome, il nome rimane invariato nonostante i cambiamenti relativi alla posizione del dato, ai vari cambiamenti del dato in sè, e a quelli del suo proprietario. L'indipendenza del nome persistente e della locazione di archiviazione, è un prerequisito per la diffusione di informazioni in maniera efficiente. Altrimenti, è difficile

sfruttare al meglio le copie cache presenti nella rete. Dal punto di vista di un utente, la persistenza dei nomi riduce i problemi di oggi di errori web *404 - file not found* dati dallo spostamento del contenuto. La persistenza del nome garantisce anche che i bookmarks rimangono validi fino a quando l'oggetto con le rispettive informazioni esiste da qualche parte nella rete. Dal punto di vista dei fornitori di contenuti, la persistenza del nome semplifica la gestione dei dati che possono essere spostati tra le cartelle e tra i diversi server come desiderato, senza preoccupparsi di cambiare il loro link o indirizzo. Più precisamente, se cambia la posizione, invece di memorizzare i reindirizzamenti corrispondenti per le nuove posizioni, è necessario solo aggiornare il corrispondente ingresso unico nel NRS [6]. La persistenza del nome permette, in sostanza l'esistenza di oggetti dinamici sotto lo stesso nome, cioè gli oggetti che cambiano nel tempo, come ad esempio le diverse versioni o uno stream live. Inoltre, assicura che il nome rimane lo stesso anche se cambia il proprietario, che è un aspetto non presente in altre soluzioni proposte precedentemente come DONA e CCN.

In NetInf, l'autenticazione del proprietario è una proprietà separata dall' auto-Certificazione. Ciò significa che si può raggiungere l'auto-certificazione senza l'autenticazione del proprietario. Tuttavia, può essere auspicabile raggiungere anche l'autenticazione del proprietario. Ma, a tal fine, la coppia di chiavi pubblica/privata associata ad un proprietario può essere diversa da quella associata ad IO.
Si distinguono due diversi tipi di autenticazione del proprietario: la prima, dove il proprietario può agire ripetutamente sull'oggetto ma può rimanere anonimo e la seconda, dove il proprietario è identificato come un nome proprio. La prima è chiamata autenticazione del proprietario, mentre la seconda si chiama identificazione del proprietario e richiede l'utilizzo di chiavi pubbliche certificate emesse da terze parti fidate, come le Certification Autority.

[6] sistema globale di NetInf Naming Resolution System

Se l'ID di un oggetto non è conosciuto come autentico, l'autocertificazione non assicura l'integrità del dato. Vale a dire, un malintenzionato può modificare il contenuto del dato, generare un nuovo ID per quel contenuto, e fornire nuove firme digitali per i dati cambiati e inserirli nei metadati. Questo nuovo oggetto, con un nuovo identificatore, apparirà autocertificato, sebbene sia fraudolento. In pratica, l'autenticità del nome non è automaticamente garantita, soprattutto se gli oggetti sono ottenuti con l'ausilio di motori di ricerca, che recuperano gli ID degli IO dagli attributi forniti. Di conseguenza, l'autenticazione del nome è una proprietà aggiuntiva prevista per essere utile nella pratica. Essa fornisce l'autenticità dell' Identificatore degli oggetti per un dato contenuto, dove il contenuto può essere specificato soltanto dall' Identificatore oppure da alcuni attributi aggiuntivi o persino nel suo complesso, con il valore hash del contenuto. Anche se un Identificatore di un oggetto è conosciuto come autentico e il contenuto del dato è auto-certificato, significa soltanto che il contenuto del dato è lo stesso come prodotto originalmente dal legittimo proprietario. Ma questo non garantisce che il contenuto del dato è attendibile in tutte le sue parti, la veridicità può essere globale o locale, ad un dato IO. La proprietà aggiuntiva di certificazione del dato, fornita da NetInf, assicura che le parti specifiche del contenuto del dato auto-certificato, sono effettivamente vere, sia globalmente che localmente.

Queste proprietà diventano difficili da raggiungere se, per esempio, si combina l'autocertificazione e la persistenza del nome per contenuti dinamici. Supportando tutte le possibili combinazioni aumenta la complessità del Framework e riduce l'usabilità. Pertanto, il Framework dovrebbe supportare un insieme di meccanismi che possono essere combinati per soddisfare i requisiti in contesti diversi. Questo consente una maggiore flessibilità e semplicità in casi particolari come i contenuti statici che non richiedono molta complessità. Inoltre deve essere in grado di poter cambiare agevolmente ad un nuovo

algoritmo, se il vecchio diventa insicuro.

Per questa serie di motivi si è cercato di effettuare una implementazione che tenga conto della praticità del framework, intesa sia come semplicità di implementazione che come complessità delle operazioni eseguite, per cui gli obiettivi sono stati in parte limitati, da aggiungere molte limitazioni invece sono dovute all'architettura propria di NetInf, in quanto ad esempio alcune proprietà devono essere garantite in momenti diversi all'utilizzo della libreria. Ad esempio, come abbiamo accennato prima, l'autenticità dell'Identificatore deve essere affettuata all'atto della registrazione dell'oggetto, per cui non vi è alcun modo di garantire questa proprietà al di fuori dell'applicazione. Nonostante ciò si è cercato di garantire tutte le proprietà possibili con l 'uso della libreria LEMS.

Capitolo 3

Sicurezza

3.1 Protocollo di Autenticazione

3.1.1 Problema della Sicurezza

Con la diffusione delle reti, e in particolare di internet, sono emerse delle funzionalità importanti che devono essere presenti nei protocolli crittografici, per garantire la confidenzialità delle comunicazioni. Una di queste funzionalità è l'autenticazione dei nodi che comunicano. In particolare, nel nostro caso, permettere ad un nuovo nodo di entrare a far parte della rete overlay XMHT (eXtensible Metadata Hash Table), fondamentale in quanto solo i nodi autenticati possono pubblicare le informazioni.

3.1.2 Crittografia

La costruzione di messaggi segreti è antica, forse, quanto la comunicazione tra gli uomini. Nata per scopi non sempre nobilissimi (perchè mascherare i messaggi se non vi è qualcosa da nascondere?) si è sviluppata in modo sistematico nella trasmissione scritta ove prende il nome di *crittografia*, etimologicamente *scrittura nascosta*; e in questa forma ha trovato dimora nei campi più vari. Dai messaggi degli amanti, alla guerra, dall'enigmistica agli intrighi diplomatici, fino alle reti di calcolatori, la crittografia ha svolto un ruolo così importante da richiedere una rigorosa impostazione metodologica. Essa cade in fatti nel dominio quasi esclusivo della matematica e in partico-

lare della complessità di calcolo e dell'algoritmica.
Oggi, con l'enorme diffusione della comunicazione elettronica e con l'importanza dei messaggi che si scambiano in questa forma, la crittografia è divenuta una disciplina critica e complessa.

La crittografia, con i suoi metodi di *cifratura*, e la *crittoanalisi*, con i suoi metodi di interpretazione, sono indissolubilmente legate tra loro e costituiscono di fatto due aspetti di una stessa scienza che è stata chiamata *crittologia*. [7][p.13]. Il problema centrale di questa scienza è schematicamente il seguente: un agente *Mitt* vuole comunicare con un altro agente *Dest* utilizzando un canale di trasmissione *insicuro*, cioè tale che altri possono intercettare i messaggi che vi transitano per conoscerli o alterarli. Per proteggere la comunicazione i due agenti devono adottare un metodo di cifratura che permetta a *Mitt* di spedire un *messaggio* m sotto forma di *crittogramma* c,incomprensibile a un ipotetico crittoanalista X in ascolto sul canale, ma di cui sia facile la decifrazione da parte di *Dest*. Se definiamo con *Msg* lo spazio dei messaggi e con *Critto* lo spazio dei crittogrammi, le operazioni di cifratura e decifrazione si definiscono formalmente come segue:

Cifratura del messaggio. Operazione con cui si trasforma un generico messaggio in chiaro m in un crittogramma c applicando una funzione $C{:}Msg \rightarrow Critto$.

Decifrazione del crittogramma. Operazione che permette di ricavare il messaggio in chiaro m a partire dal crittogramma c applicando una funzione $D{:}Critto \rightarrow Msg$.

Una possibile classificazione dei metodi crittografici, detti *cifrari*, è legata al grado di segretezza del metodo, che a sua volta dipende dall'uso cui è destinato il cifrario. Vi sono cifrari per uso ristretto, impiegati principalmen-

te per comunicazioni diplomatiche o militari, in cui le funzioni di cifratura C e D sono tenute segrete in ogni loro aspetto. Tuttavia, è impossibile attuare questo metodo crittografico ad una crittografia di massa, così nascono i *cifrari per uso generale*, fondati non sulla segretezza delle funzioni C e D, le quali sono addirittura pubblicamente note, ma sull'uso di una *chiave segreta* k diversa per ogni coppia di utenti. Molti dei cifrari in uso oggi sono a chiave segreta e vengono detti *cifrari simmetrici*. Naturalmente occorre porre molta attenzione nella scelta della chiave e nell'operazione di scambio della chiave stessa tra *Mitt* e *Dest*, poichè una intercettazione in questa fase pregiudicherebbe irrimediabilmente il sistema.

Nel 1976 *Diffie e Hellman*, e indipendentemente *Merkle*, introdussero un nuovo concetto che avrebbe rivoluzionato il modo di concepire le comunicazioni segrete: *i cifrari a chiave pubblica*. Nei *cifrari simmetrici* visti sinora la chiave di cifratura è uguale a quella di decifrazione (o comunque ciascuna può essere facilmente calcolata dall'altra), ed è nota solo ai due partner che la scelgono di comune accordo e la mantengono segreta. Nei cifrari a chiave pubblica, o *asimmetrici*, le chiavi di cifratura e di decifrazione sono completamente diverse tra loro. Esse sono scelte dal destinatario che rende pubblica la chiave di cifratura k[pub], che è quindi nota a tutti, e mantiene segreta la chiave di decifrazione k[priv] che è quindi nota soltanto a lui. Esiste quindi una coppia di chiavi pubblica e segreta $\langle k[pub], k[priv] \rangle$ per ogni utente. La cifratura di un messaggio m da inviare a *Dest* è eseguita da qualunque mittente come c = $C(m, k[pub])$, ove sia la chiave k[pub] che la funzione di cifratura C sono note a tutti. La decifrazione è eseguita d *Dest* come m = $D(c, k[priv])$, ove D è la funzione di decifrazione anch'essa nota a tutti, ma k[priv] non è disponibile agli altri che non possono quindi ricostruire m. L'appellativo di asimmetrici assegnato a questi cifrari sottolinea i ruoli completamente diversi svolti da *Mitt* e *Dest*, in contrapposizione ai ruoli intercambiabili che essi hanno nei cifrari simmetrici, ove condividono

la stessa informazione (cioè la chiave) segreta.
Per funzionare correttamente, il processo di cifratura e decifrazione deve soddisfare alcune proprietà:

1. Per ogni possibile messaggio m si ha: $D(C(m, k[pub]), k[priv]) = m$. Ossia *Dest* deve avere la possibilità di interpretare qualunque messaggio che gli altri utenti decidano di spedirgli.
2. La sicurezza e l'efficienza del sistema dipendono dalle funzioni *CeD*, e dalla relazione che esiste tra le chiavi k[priv] e k[pub] di ogni coppia.

Più esattamente:

(a) la coppia $\langle k[pub], k[priv] \rangle$ è computazionalmente facile da generare, e deve risultare praticamente impossibile che due utenti scelgano la stessa chiave;

(b) dati m e k[pub], è computazionalmente facile per il mittente calcolare il crittogramma c = $C(C(m, k[pub])$;

(c) dati c e k[priv], è computazionalmente facile per il destinatario calcolare il messaggio originale m = $D(c, k[priv])$;

(d) pur conoscendo il crittogramma c, la chiave pubblica k[pub], e le funzioni *CeD*, è computazionalmente difficile per un crittoanalista risalire al messaggio m.

La proprietà 1 garantisce la correttezza del processo complessivo di cifratura e decifrazione. La proprietà 2 esclude che due utenti possano avere le stesse chiavi imponendo in pratica che la generazione delle chiavi sia casuale. Le proprietà 2b e 2c assicurano che il processo possa essere di fatto adottato. La proprietà 2d garantisce la sicurezza del cifrario.
Da tutto questo deriva che C deve essere una funzione *one way* [1], cioè fa-

[1] vedi 3.2.4 Funzioni hash one way

cile da calcolare e difficile da invertire, ma deve contenere un meccanismo segreto detto emphtrap-door che ne consenta la facile invertibilità solo a chi conosca tale meccanismo (chiave privata k[priv]).
In questa tesi vengono utilizzati i *cifrari simmetrici* per l'implementazione del protocollo di autenticazione e come vedremo più avanti verranno utilizzati i *cifrari asimmetrici* per la realizzazione della libreria LEMS, che sfrutta appieno le tecniche di cifratura asimmetrica.

3.1.3 Scelta dell'Algoritmo

Tra i cifrari a chiave simmetrica più conosciuti come il DES, RC5, IDEA e AES, è stato scelto il cifrario AES per l'implementazione del protocollo di autenticazione.
In crittografia, l'Advanced Encryption Standard (AES), conosciuto anche come Rijndael, di cui più propriamente ne è una specifica implementazione, è un algoritmo di cifratura a blocchi utilizzato come standard dal governo degli Stati Uniti d'America [14]. Data la sua sicurezza e le sue specifiche pubbliche si presume che in un prossimo futuro venga utilizzato in tutto il mondo come è successo al suo predecessore, il Data Encryption Standard (DES) che ha perso poi efficacia per vulnerabilità intrinseche. AES è stato adottato dalla National Institute of Standards and Technology (NIST) e dalla US FIPS PUB nel novembre del 2001, dopo 5 anni di studi, standardizzazioni e selezione finale tra i vari algoritmi proposti. L'algoritmo scelto è stato sviluppato da due crittografi belgi, Joan Daemen e Vincent Rijmen, che lo hanno presentato al processo di selezione per l'AES con il nome di Rijndael, derivato dai nomi degli inventori. [14]
Il NIST ha giustificato la scelta sostenendo che Rijndael è il cifrario che offre la migliore combinazione di sicurezza, efficienza, realizzabilità hardware e software, e flessibilità.

Sicurezza del cifrario

La National Security Agency (NSA) segnalava che tutti i finalisti del processo di standardizzazione erano dotati di una sicurezza sufficiente per diventare l'AES, ma che fu scelto il Rijndael per via della sua flessibilità nel trattare chiavi di lunghezza diversa, per la sua semplice implementazione in hardware e in software e per le sue basse richieste di memoria che ne consentono un'implementazione anche in dispositivi con scarse risorse come le smart card. Durante il 2004 non si sono verificate forzature dell'AES. L'AES può essere utilizzato per proteggere le informazioni secretate. Per il livello SECRET è sufficiente una chiave a 128 bit mentre per il livello TOP SECRET si consigliano chiavi a 192 o 256 bit. Questo significa che per la prima volta il pubblico ha accesso ad una tecnologia crittografica che NSA ritiene adeguata per proteggere i documenti TOP SECRET. Si è discusso sulla necessità di utilizzare chiavi lunghe (192 o 256 bit) per i documenti TOP SECRET. Alcuni ritengono che questo indichi che l'NSA ha individuato un potenziale attacco che potrebbe forzare una chiave relativamente corta (128 bit), mentre la maggior parte degli esperti ritiene che le raccomandazioni della NSA siano basate principalmente sul volersi garantire un elevato margine di sicurezza per i prossimi decenni contro un potenziale attacco esaustivo. La maggior parte degli algoritmi crittografici viene forzata riducendo il numero di round. L'AES effettua 10 round per la chiave a 128 bit, 12 round per la chiave a 192 bit e 14 round per la chiave a 256 bit. I migliori attacchi sono riusciti a forzare l'AES con 7 round e chiave di 128 bit, 8 round e chiave di 192 bit e 9 round e chiave di 256 bit [14]. Alcuni crittografi hanno fatto notare che la differenza tra i round effettuati dall'AES e quelli massimi prima che l'algoritmo non sia più forzabile è ridotta (specialmente con chiavi corte). Questi temono che miglioramenti nelle tecniche di analisi possano permettere di forzare l'algoritmo senza verificare tutte le chiavi. Attualmente, una ricerca esaustiva è impraticabile: la chiave

a 128 bit produce $3.4 * 10^{38}$ combinazioni diverse. Uno dei migliori attacchi a forza bruta è stato svolto dal progetto distributed.net su una chiave a 64 bit per l'algoritmo RC5; l'attacco ha impiegato quasi 5 anni, utilizzando il tempo libero di migliaia di CPU di volontari [14]. Anche considerando che la potenza dei computer aumenta nel tempo, servirà ancora molto tempo prima che una chiave da 128 bit sia attaccabile con il metodo forza bruta. Un altro dubbio riguardante l'AES deriva dalla sua struttura matematica. A differenza della maggior parte degli algoritmi a blocchi, per l'AES esiste un'approfondita descrizione matematica. Sebbene non sia mai stata utilizzata per condurre un attacco su misura, questo non esclude che in futuro questa descrizione non venga utilizzata per condurre un attacco basato sulle sue proprietà matematiche. Nel 2002 l'attacco teorico chiamato attacco XSL annunciato da Nicolas Courtois e Josef Pieprzyk ha mostrato un potenziale punto debole dell'AES (e di altri cifrari) [14]. Sebbene l'attacco sia matematicamente corretto, è impraticabile nella realtà, per via dell'enorme tempo macchina richiesto per metterlo in pratica. Miglioramenti nell'attacco hanno ridotto il tempo macchina richiesto e quindi, in un futuro, questo attacco potrebbe diventare attuabile. Ultimamente, alcuni esperti hanno fatto delle osservazioni agli autori dell'attacco. Sembra che abbiano commesso degli errori teorici e che, in realtà, le loro stime siano ottimistiche [14]. Allo stato attuale, la reale pericolosità dell'attacco XSL è un punto interrogativo. Comunque, attualmente, l'AES è considerato un algoritmo veloce, sicuro e gli attacchi, fino ad ora presentati, si sono rivelati degli interessanti studi teorici ma di scarsa utilità nella pratica. In data 1 Luglio 2009 è stato pubblicato un attacco correlato alla chiave migliore del metodo forza bruta su tutti i round di AES-256 e AES-192 [14]. L'attacco in questione risulta comunque, per stessa ammissione degli autori (come chiarito nelle conclusioni dello studio), essere ancora solo teoricamente realizzabile e non dovrebbe influire in alcun modo sulla sicurezza delle odierne applicazioni che fanno uso di questo cifrario. Secondo Bruce Schneier comunque, questa scoperta potrebbe

influire negativamente sulla scelta di AES come blocco costitutivo del nuovo algoritmo di hash in fase di definizione SHA-3 [14].[2]

3.1.4 Criticità della chiave

Come evidenziato precedentemente, la scelta della chiave è un passo cruciale per un sistema crittografico, legato alla sua lunghezza in modo da non essere forzata facilmente [3] e soprattutto alla sua intercettazione che pregiudicherebbe irrimediabilmente il sistema.
Nel protocollo di autenticazione implementato è stata utilizzata una chiave di 128 bit, utile a garantire la robustezza per informazioni SECRET come evidenziato sopra. Inoltre la lunghezza in bit della chiave può facilmente essere aumentata a 192 o 256 bit a seconda del livello di segretezza che si vuole ottenere nell'overlay.
Si potrebbe pensare all'utilizzo di un cifrario a chiave pubblica per lo scambio della chiave segreta, ma questo comporterebbe un ulteriore incremento di coppie di chiavi pubbliche e private per ogni nodo, per cui è stato ipotizzato che sia il nodo che richiede la *JOIN*, che i nodi dell'overlay, siano già a conoscenza della chiave, passatagli prima in altri modi, con l'utilizzo di canali sicuri o meglio ancora uno scambio di persona.

3.1.5 Protocollo

La salvaguardia ed il controllo dell'informazione digitale sulle reti di comunicazione sono realizzate mediante l'impiego di protocolli crittografici. Un protocollo (di comunicazione) è un algoritmo che comporta uno scambio di messaggi tra due o più parti; esso è definito da una sequenza finita di

[2] una descrizione approfondita del cifrario Rijndael è disponibile nel sito Web http://aes.nist.gov/aes, insieme a un rapporto tecnico del NIST che descrive minuziosamente l'intero processo di selezione dell'AES[7]
[3] facilmente inteso come computazionalmente facile

passi che specificano in modo univoco le azioni richieste dalle parti per realizzare determinati obiettivi. Un protocollo crittografico è un protocollo che utilizza tecniche crittografiche al fine di realizzare degli obiettivi di sicurezza. Oltre all'autenticazione, obiettivo principale del protocollo, si aggiungono altre proprietà importanti relative ai singoli messaggi trasmessi, quali la riservatezza (o confidenzialità), l'autenticità, l'unicità e la non-ripudiabilità.

Il protocollo di autenticazione proposto ha il seguente schema:

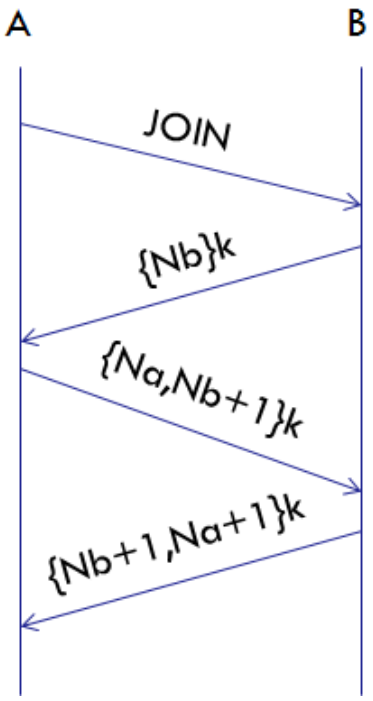


Fig. 3.1: schema del protocollo di autenticazione di un generico nodo

3.1.6 Descrizione Schema

Lo schema sopra è composto da 4 passi coincidenti con 4 tipi di messaggi scambiati tra il generico nodo che richiede la Join e il nodo bootstrap dell'overlay.

Chiamiamo Mitt il nodo che effettua la richiesta e Dest il nodo bootstrap.s

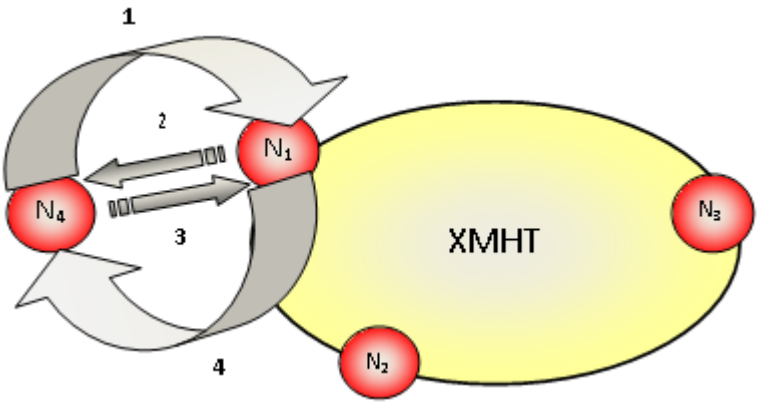


Fig. 3.2: descrive i messaggi scamabiati dal protocollo

Mitt (1): invia un messaggio di *REQUEST_JOIN* a Dest, il messaggio viene cifrato con la chiave segreta a conoscenza di Mitt.

Dest (2): elaborando la richiesta di Join, genera una quantità fresca *nonce* [4], cifra il messaggio con la stessa chiave segreta, e invia a Mitt il messaggio *ACK_REQUEST_JOIN* contenente il nonce.

Mitt (3): ricevendo il nonce da Dest, genera un suo nonce e invia un messaggio di tipo *SECURE_JOIN* contenente il suo valore di nonce e il nonce ricevuto da Dest incrementato di uno, anche questo messaggio cifrato dalla chiave segreta.

Dest (4): controlla i valori nonce ricevuti da Mitt, in particolare controlla se il valore inviato precedentemente corrisponde a quello ricevuto incrementato di uno. Invia un ultimo messaggio di conferma *JOIN_REPLY* contenente il suo e il nonce di Mitt anch'esso incrementato di uno, inoltre aggiorna la routing table interna aggiungendo l'Identificatore di Mitt.

Mitt: il mittente a questo punto riceve la conferma della richiesta e verificando i relativi nonce si accerta dell'avvenuto Join alla rete overlay

[4]contrazione di only nonce: sono numeri random inediti, freschi

XMHT.

Il protocollo, come abbiamo descritto prima, è basato sulla crittografia simmetrica (chiave condivisa); permette di stabilire una connessione cifrata con l'uso una chiave segreta precedentemente assegnata da una autorità in qualche forma. Inoltre il protocollo utilizza degli indentificatori di freschezza nonce [5]; il nonce è una componente numerica casuale e impredicibile, inedita in ogni precedente sessione del protocollo, generata per smascherare eventuali repliche, da parte di un attaccante, di messaggi intercettati in sessioni precedenti del protocollo.
Gli obiettivi che si intendono raggiungere con questo protocollo di autenticazione sono descritti di seguito.

Obiettivi

Il protocollo rispetta i seguenti requisiti:

- Segretezza delle comunicazioni
- Autenticazione reciproca delle parti (paternità del messaggio)
- integrità
- Non ripudiabilità

Difesa dagli attacchi

i Attacco replay: viene impedito con l 'utilizzo della quantità fresca nonce, il valore viene generato da un partner, mentre l'altro partner lega un'informazione criptata a questo numero per garantire la freschezza.

ii Attacco di tipo interleaving: viene impedito concatenando i messaggi del protocollo.

[5] contrazione di only-once, sono numeri random inediti, freschi

iii Attacco man in the middle: viene impedito autenticando reciprocamente le due parti, evitando all'avversario di impersonare il sistema partner.

3.1.7 Logica BAN

Le tecniche consistenti nel fare riferimento ad una serie di principi teorici e ad analisi umane per la verifica della sicurezza di un protocollo di comunicazione risultano del tutto insoddisfacenti nell'individuare potenziali nuovi attacchi.

Per molto tempo i potocolli sono stati progettati ed analizzati utilizzando quelle che ora vengono chiamate in letteratura *tecniche informali*, consistenti nel fare riferimento ad una serie di principi teorici e ad analisti umani per la verifica della sicurezza di un protocollo. Le tecniche informali hanno permesso in vari casi di determinare se un dato protocollo fosse suscettibile o meno di violazione per effetto di attachi noti, ma risultano del tutto insoddisfacenti per la verifica della sicurezza dei protocolli relativamente a nuovi attacchi. Ciò deriva, in primo luogo, dall'impossibilità per un analista umano di prevedere tutti i possibili piani d'attacco, i quali possono includere l'esecuzione simultanea di parecchie sessioni del protocollo.

In alternativa alle suddette tecniche sono state quindi concepiti altri approcci, i cosiddetti *metodi formali*, che permettono l 'analisi di protocolli, anche complessi, con metodi matematici. Grazie all'impiego di questi nuovi metodi sono stati rilevati alcuni difetti in protocolli già dichiarati ampiamente sicuri.

La verifica del protocollo realizzato consiste nel dimostrare che il protocollo realizza effettivamente gli obiettivi di sicurezza definiti e perseguiti in fase di progetto. Gli obiettivi del seguente protocollo sono la freschezza delle informazioni scambiate in fase di autenticazione, nonchè l'autenticazione stessa. Questi obiettivi sono raggiunti attraverso l'analisi logica BAN (Burrows-Abadi-Needham) [23]. Questo tipo di metodo formale, ha il preciso intento

di minimizzare l'errore umano utilizzando dimostrazioni brevi.
Di seguito viene descritto il formalismo BAN per capire meglio la dimostrazione seguente del protocollo realizzato.

Formalismo BAN

- $P \mid\equiv X$ **P believes X:** P si comporta come se X fosse vero
- $P \mid$ **P sees X:** P ha ricevuto un messaggio che contiene X, nel passato o in questa esecuzione del protocollo; P può leggere X e ripeterlo
- $P \mid\sim X$ **P once said X:** P ha inviato un messaggio che contiene X; P believed X quando lo inviò
- $P \Rightarrow X$ **P coontrols X:** P è un autorità su X e bisogna fidarsi a questo riguardo
- $\sharp(X)$ **X è fresh**
- $P \overset{k}{\longleftrightarrow} Q$ **k è una chiave condivisa tra P e Q**
- $P \overset{k}{\rightleftharpoons} Q$ **k è un segreto condiviso tra P e Q**
- $\overset{k}{\mapsto} P$ **k è la chiave pubblica di P**
- $\langle X \rangle_y$ **X è combinato con y (segreto)**
- $\{X\}_y$ **X è stato cifrato con k**

Di seguito viene descritta l'analisi del protocollo BAN.

3.1.8 Dimostrazione (BAN)

Descrizione Formale del Protocollo

PROTOCOLLO REALE:

Step 1: $A \rightarrow B$ $\quad JOIN$

Step 2: $B \rightarrow A$ $\quad E_k(Nb)$

Step 3: $A \rightarrow B$ $\quad E_k(Na, Nb+1)$

Step 4: $B \rightarrow A$ $\quad E_k(Na+1, Nb+1)$

PROTOCOLLO IDEALIZZATO:

Step 1: $A \rightarrow B : JOIN$

Step 2: $B \rightarrow A : \{Nb\}_k$

Step 3: $A \rightarrow B : \{Na, Nb+1\}_k$

Step 4: $B \rightarrow A : \{Na+1, Nb+1\}_k$

IPOTESI :

$A \mid\equiv A \overset{k}{\longleftrightarrow} B$

$B \mid\equiv A \overset{k}{\longleftrightarrow} B$

$A \mid\equiv \sharp(N_a)$

$B \mid\equiv \sharp(N_b)$

$A \mid\equiv B \mid\Rightarrow N_b$

$B \mid\equiv A \mid\Rightarrow N_a$

TESI:

$A \mid\equiv B \mid\equiv N_b$

$B \mid\equiv A \mid\equiv N_a$

PROGRESS:

M 2: $\dfrac{A \mid\equiv A \overset{k}{\longleftrightarrow} B, A \triangleleft \{Nb\}_k}{A \mid\equiv B \mid\sim \{Nb\}}$ MMR

M 3: $\dfrac{B \mid\equiv A \overset{k}{\longleftrightarrow} B, B \triangleleft \{Na, Nb+1\}_k}{B \mid\equiv A \mid\sim \{Na, Nb+1\}}$ MMR

$\dfrac{B \mid\equiv \sharp(Nb), B \triangleleft \{Na, Nb+1\}_k}{B \mid\equiv A \mid\equiv Na}$ NVR

M 4: $\dfrac{A \mid\equiv A \overset{k}{\longleftrightarrow} B, A \triangleleft \{Na+1, Nb+1\}_k}{A \mid\equiv B \mid\sim \{Na+1, Nb+1\}}$ MMR

$\dfrac{A \mid\equiv \sharp(Na), A \triangleleft \{Na+1, Nb+1\}_k}{A \mid\equiv B \mid\equiv Nb}$ NVR

MMR: *Message Meaning Rule* **NVR:** *Nonce Verification Rule*

3.1.9 Implementazione del protocollo

Linguaggio e Librerie utilizzate

Il protocollo è stato realizzato in C++, la scelta del linguaggio è stata forzata dal fatto che l'overlay xmht è stato scritto in C++. Anche l'uso delle librerie utiizzate sono state determinate dall'overlay. In particolare è stata utilizzata la libreria Cryptopp 5.6.0 [6]per la cifratura e la decifrazione dei messaggi sopra descritti. Cryptopp è una libreria open source che fornisce gli strumenti di cifratura grazie a degli schemi e algoritmi crittografici di rilievo come il DES, Ttriple-DES, IDEA e AES, quest'ultimo scelto per l'implementazione del protocollo di autenticazione. La libreria supporta sistemi operativi con architettura di 32-64 bit, e può essere usata in sistemi Windows, Linux, Apple e Solaris.

Inoltre è stata utilizzata la libreria Protocol Buffer [7], per la serializzazione dei dati nello scambio di messaggi. Le Protocol Buffer rappresentano il formato di interscambio (data interchange format) utilizzato internamente da google. Questo formato viene descritto come un formato per serializzare in maniera estensibile, indipendente dal linguaggio e dalla piattaforma strutture dati da utilizzare in protocolli di comunicazione, archiviazione di dati e altro. Protocol Buffers permette agli sviluppatori di definire strutture dati in uno speciale linguaggio che, successivamente alla compilazione, produrrà classi (per Java, C++, Python) rappresentanti tali strutture; il codice prodotto è fortemente ottimizzato ed ogni classe include metodi set() e get() pronti all'uso.

Inizialmente si ricorreva anche all'utilizzo delle librerie Boost [8]. Le Librerie C++ Boost sono una collezione di librerie open source che estendono le funzionalità del C++. Molte di esse sono licenziate sotto la Boost Software License in modo da poter essere utilizzate sia in progetti open source che

[6]http://www.cryptopp.com/
[7]abbreviati protobuf: http://code.google.com/p/protobuf/
[8]http://www.boost.org/

closed source. Per assicurare efficienza e flessibilità, Boost fa un estensivo utilizzo della programmazione basata su template, e quindi sulla programmazione generica e metaprogrammazione. Con l'utilizzo di queste lbrerie l'overlay si proiettava ad un sistema di tipo multipiattaforma, ma in seguito questa libreria è stata tolta, in quanto veniva utilizzata soltanto per la gestione dei socket e soprattutto appesantiva molto l'overlay. Questa scelta ha snellito sicuramente l'overlay, ma per eseguirlo su altri sistemi bisogna effettuare dei wrapper adeguati per l 'utilizzo dei socket. [9].
Tutte le funzionalitá relative al protocollo sono state aggiunte al progetto XMHT all'interno della classe Autenticator.cpp.

protected ByteArray_t <16> key_ chiave di autenticazione per la cifratura e la decifratura dei messaggi

protected ByteArray_t <16>cipherText_ testo cifrato da inviare

public uint32_t nonce_node_ nonce generato in maniera *random* del nodo corrente

public uint32_t nonce_bootstrap_ nonce generato in maniera *random* del nodo bootstrap

Funzioni

Le funzioni della classe sono:

byte* GetKey(void) ritorna la variabile *key_* di tipo byte* che rappresenata la chiave di autenticazione del nodo.

byte* GetText(void) ritorna la variabile Text_ di tipo byte * che con-

[9]esempio in sistemi Windows si usano gli Winsock

tiene il testo cifrato del nodo.

word32 GetNonce(void) ritorna il nonce del nodo.

word32 GetNonceBootstrap(void) ritorna il nonce del nodo bootstrap.

void SetNonce(word32 nonce) assegna alla variabile *nonce_node_* il valore di nonce passatogli come parametro.

void SetNonceBootstrap(word32 nonce) assegna alla variabile *nonce_bootstrap_* il valore di nonce passatogli come parametro.

void SetText(byte Text[]) scrive in *Text_* il valore dell'array Text passatogli come parametro.

void SetKey(byte key[]) scrive in *Key_* il valore dell'array key passatogli come parametro.

void GenerateNonce(void) genera un valore random compreso tra 100 e 10000 che rappresenta il nonce del nodo e lo assegna alla variabile *nonce_node_*.

void StoreValueInStr(unsigned char* MessageRef, unsigned long ulOffset, unsigned long ulValue, unsigned long ulNbrOfBits) si occupa di preparare la parte del messaggio che contiene il valore dei nonce, inserendo il testo in bit nella posizione passatagli come parametro. Inoltre si assicura che i valori vengono inseriti nello stesso ordine per evitare problemi relativi all' allineamento dei bit dovuti al tipo di macchina (BIG-ENDIAN/LITTLE-ENDIAN).

unsigned long GetValue(unsigned char* prtBase,ulong offset,unsigned char size) si occupa dello spacchettamento del messaggio, ovvero della parte del messaggio realtivo ai nonce.

Come abbiamo visto in precedenza, per lo scambio dei messaggi l'overlay XMHT usa la libreria Protocol Buffers, efficace per la serializzazione e deserializzazione dei dati, molto adatto per lo scambio di dati XML. Le funzioni che si occupano della serializzazione e della serializzazione dei dati sono le seguenti:

virtual bool toSerializedMessage(std::string & serializedMessage) const serializza l'intero messaggio scambiato dai nodi dell'overlay.

virtual bool fromSerializedMessage(const std::string & serializedMessage) deserializza il messaggio ricevuto prima di interpretarlo.

3.2 Libreria LEMS

L'obiettivo della libreria LEMS [10], è quello di fornire uno strato di supporto tra l'applicazione e le api dell'overlay xmht, in questo modo vengono filtrate le richieste di pubblicazione e di recupero dei metadati, garantendo che le richieste di pubblicazione siano autorizzate dai legittimi proprietari dell'oggetto e i metadati ricevuti siano integri e autentici, ovvero corrispondenti all'Identificatore relativo all'oggetto.

3.2.1 Architettura Lems

Di seguito si riporta un grafico esplicativo dell'Architettura LEMS, e del suo posizionamento nell'architettura generale di NetInf.

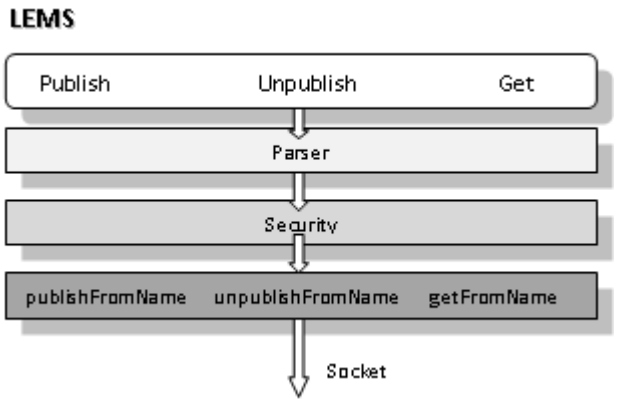


Fig. 3.3: Interfaccia LEMS

3.2.2 Scelta del Linguaggio

La libreria è stata realizzata in Java, la scelta del linguaggio è dovuta soprattutto alla versatilità, efficienza, portabilità della piattaforma e sicurezza, la tecnologia Java è ideale per il network computing.
Oltre alla semplicità di implementazione e all'efficienza del garbage collector per la gestione automatica della memoria, java mette a disposizione un

[10] Library Extensible Metadata Security

ricco insieme di package che sono molto utili per la gestione della sicurezza. In particolare è stato usato il package *java.security* [15], mentre per un eventuale cifratura occorre il package *javax.crypto* [16].

3.2.3 Strutture Dati

Le strutture dati utilizzate sono le due classi *IdentificatorS* e *MetadataS*. Di seguito la descrizione dettagliata della struttura dell'Identificatore dell'oggetto, del suo metadato e successivamente introdurremo anche la struttura del suo owner, che come vedremo avrà la stessa struttura del nome dell'Identificatore dell'IO.

Identificatore

l'Identificatore ha la seguente struttura

$$\text{TYPE}_{IO} \mid A_{IO} = HASH(PK_{IO}) \mid L_{IO}$$

- Il campo Type definisce il tipo di Information Object, se è statico o dinamico, inoltre definisce il tipo di owners nel caso sia anonimo.
- Il campo A relativo all'autenticazione lega l'Identificatore alla chiave pubblica PK. La funzione *hash* è una funzione hash crittografica, la quale deve essere one-way [11] e resistente alle collisioni. La funzione hash serve soltanto a ridurre la lunghezza in bit della chiave pubblica PK. La chiave pubblica PK è quindi associata ad ogni singolo IO. Verificare se una chiave pubblica è consistente con un determinato ID (o viceversa) basta confrontare il suo valore hash con A. Si presume che PK è generata secondo il sistema crittografico a chiave pubblica, dove il segreto è dato dalla chiave privata SK nota soltanto al legittimo proprietario. Di conseguenza, un proprietario di un IO è definito

[11] vedere 3.2.4 Funzioni hash one way

come una qualsiasi entità che conosce la chiave privata SK o qualsiasi altra chiave segreta SK' autorizzata da SK, dove SK o SK' sono usate per l'autenticazione firmando i dati da essere autenticati. Questo è possibile usando l 'algoritmo crittografico della firma digitale DSAlg, dove prima viene fatto l 'hash del dato usando una funzione hash h, possibilmente diversa da quella usata per calcolare il campo di autenticazione A. Il campo A è obbligatorio tranne che per gli IO con owners statici o anonimi.

- Il campo L contiene una numero arbitrario di tag di identificazione associato ad un IO. Nel nostro caso il campo L viene utilizzato prettamente per l'autocertificazione nel caso un Information Object sia di tipo statico.

TYPE

- **typeIO** specifica se un IO è statico, dinamico o anonimo [12]

- **typeHashL** solo se il tipo di IO è statico specifica la funzione hash utilizzata per l'hashing del contenuto statico nel campo L

- **typeHashA** specifica la funzione hash utilizzata per l'hashing della chiave pubblica PK dell'IO nel campo A

Campo A

- **autenticator** contiene il valore hash della chiave pubblica dell'Information Object [13]

[12] se è statico il contenuto dell'oggetto rimane invariato, se dinamico può cambiare nel tempo, esempio nel caso di uno streaming live, invece se anonimo è liberamente pubblicabile, cioè, è consentita la registrazione/cancellazione e pubblicazione nel nostro caso senza autorizzazione da parte di alcun proprietario

[13] Il campo A relativo all'autenticazione lega l'Identificatore alla chiave pubblica PK. La funzione hash è una funzione hash crittografica, la quale deve essere one-way e resistente alle collisioni. La funzione hash serve soltanto a ridurre la lunghezza in bit della chiave pubblica PK. La chiave pubblica PK è quindi associata ad ogni singolo IO, Il campo A è obbligatorio tranne che per gli IO con owners statici o anonimi

Campo L

- **tagL** contiene il valore hash del contenuto di un IO per l'autenticazione statica di un IO [14]

Si noti che il tipo precisa le funzioni hash utilizzate nell'ID e quindi anche in questo caso è utile per almeno due ragioni. In primo luogo, l'utilizzo di un numero limitato di funzioni hash considerato sicuro dalla comunità cripto può quindi essere forzato. In secondo luogo, nel caso in cui una funzione hash utilizzata diventa insicura, il Tipo può essere sfruttato dall' NRS per riconoscere automaticamente gli ID che utilizzano questa funzione hash per renderli non validi e rimuoverli dal NRS. In ogni caso, cambiando le funzioni hash utilizzate nell'ID si deve cambiare l'ID stesso.

IdentificatoreS.java

```
package Lems;

import java.io.IOException;
import java.io.StringReader;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
```

[14]Il campo L contiene una numero arbitrario di tag di identificazione associato ad un IO, con una struttura fissa o variabile specificata dal campo Tipo. Questi tag possono essere correlati al contenuto dei dati e/o proprietari, o essere numeri seriali oppure numeri random

```
import org.xml.sax.SAXException;
public class IdentifierS {

/** crea l'identificatore da passare come
  * argomento per la publish dell'oggetto */
public String IdentifierS_;

/** definisce il tipo di Information Object
 *  se IO Statico , Dinamico o Anonimo */
public String typeIO_ = new String();

/** tipo funzione hash per A */
public String typeHashA_ = new String();

/** tipo funzione hash di L */
public String typeHashL_ = new String();

/** definisce il campo Autenticator A ,
 *  comprende il valore hash della chiave pubblica PK
 *  dell'Information Object A = HASH(PK) */
public String autenticator_ = new String();

/** definisce il campo L dell'identificatore */
public String tagL_ = new String();

/* definisce il contenuto del dato,
 * utilizzata come impronta del dato
 * per garantire l'integrita'ă
 * e la confidenzialita'ădello stesso */
public byte[] s_certData_;
```

```
public IdentifierS(){}

/** funzione che si occupa di
  * parsare l'identificatore sotto
  * forma di XML, estrapolando tutte
  * le informazioni necessarie
  * per garantire le proprieta'
  * crittografiche richieste */
public void ParsIdentificator(String xml_identificatore)
{....}
```

Metadato

Il Metadato ha la seguente struttura

$$\text{METADATA}_{Type}|METADATA$$

Il Metadato ha la stessa struttura dei nomi dell'Identificatore, per cui è costituito da due parti principali: la prima $METADATA_{Type}$ contiene *il locatore*, ovvero le informazioni relative al routing del *DATO,(il suo indirizzo IP, il protocollo e la porta su cui poter reperire il dato*, il metadato infatti oltre ad offrire proprietà crittografiche è fondamentale per il recupero e il routing del dato stesso. La seconda parte *METADATA* contiene le informazioni necessarie per i meccanismi di sicurezza. I metadati si distinguono in due tipi, i metadati attributo e i metadati di sicurezza, tutte e tre i tipi di metadati possono essere estesi a piacimento per ulterioni caratteristiche aggiuntive al framework. [15]

[15]ad esempio si può pensare di localizzare l'Information Object con le sue coordinate geografiche al posto del suo indirizzo IP; le coordinate geografiche sono già presenti nei metadati, nonostante in questa libreria non vengono utilizzate per i meccanismi di sicurezza, possono comunque essere utili per molte applicazioni future.

- I metadati attributo includono le coppie attributo-valore che descrivono il nome del dato. In particolare, gli attributi possono essere direttamente derivati dal contenuto del dato (ad esempio, descrizione semantica del contenuto del dato) o può fornire ulteriori caratteristiche di un IO (ad esempio, tempo di produzione, scadenza, numero di versione, numero di segmento, l'ID del proprietario, ecc). Questo tipo di Metadato non è richiesto dal framework, ma può, essere necessario come vedremo nel seguito per garantire alcune proprietà di sicurezza del framework

- I metadati di sicurezza comprendono tutti i metadati che sono necessari per svolgere le funzioni di sicurezza integrate nel framework NetInf, ad esempio, tutti i metadati necessari per effettuare l'autocertificazione, l'autenticazione e l'identificazione del proprietario, l'autenticazione del nome, e la certificazione del dato. Questa parte specifica la funzione hash h e l'algoritmo DSAlg utilizzato per la firma digitale. Esso inoltre include tutta la chiave pubblica PK di IO (non solo il valore hash come in ID).

ATTRIBUTE

- **gps** contiene le coordinate gps dell'Information Object [16]

- **semantic** può contenere una descrizione semantica dell'oggetto [17]

- **identificatorO** include l'Identificatore del proprietario

SECURITY

[16] le coordinate gps non sono state considerate ai fini della sicurezza tuttavia sono molto utili per gli sviluppi futuri, garantisce l'estensibilità della libreria per applicazioni centrate sulle coordinate geografiche dell'oggetto

[17] questo campo non viene utilizzato nella nostra libreria ma può essere molto utile ai fini della ricerca di un Information Object da parte delle applicazioni

- **sign** contiene l'hash del metadato firmato con la chiave privata SK corrispondente alla chiave pubblica PK dell'Information Object
- **sign_O** contiene l'hash del metadato firmato con la chiave privata SK del proprietario dell'Information Object
- **pk** chiave pubblica dell'Information Object
- **pk_O** chiave pubblica del proprietario dell'Information Object
- **typeHash** contiene la funzione hash utilizzata prima della firma digitale
- **typeDSAlg** contiene il tipo di algortimo per la firma digitale dell'oggetto

MetadataS.java

```
package Lems;

import java.io.IOException;
import java.io.StringReader;
import java.io.StringWriter;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
```

```
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import xmhtapi.Global.Coordinate;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

public class MetadataS {

public MetadataS(){}

/** metadata contiene la stringa
  * xml senza il campo f_sCertData
  * in modo da poter verificare
  * la firma digitale dei metadati
  * contenuti nei metadati associativi */
public  String metadata_ = new String();

/** contiene le informazioni dei
  * metadati attributo, tipo le
  * le coordinate gps, decrizione
  * semantica dell'oggetto IO,
  * e l'identificatore dell'owner,
  * quet'ultimo utile per la
  * proprietà di Autenticazione
```

```
  * e/o Identificazione del proprietario */
Coordinate gps_ = new Coordinate();
String semantic_ = new String();
String identificatorO_ = new String();

/** contiene la firma di sCertData
  * (nel nostro caso il metadato),
  * con la chiave privata SK di IO,
  * utile per verifica
  * della proprietà di Autocertificazione */
public String sign_ = new String();

/** contiene la firma di sCertData
  * (nel nostro caso il metadato),
  * con la chiave privata SK di owner,
  * utile per verifica
  * della proprietà di Autocertificazione */
public String sign_O_ = new String();

/** chiave pubblica dell'oggetto IO */
public String pk_ = new String();

/** chiave pubblica dell'owner dell'oggetto IO */
public String pk_O_ = new String();

/** tipo di funzione hash one way utilizzata */
public String typeHash_ = new String();

/** tipo di algoritmo utilizzato per la firma digitale */
public String typeDSAlg_= new String();
```

```
/**
 * @throws TransformerException
 * @throws IOException
 * @throws NoSuchAlgorithmException
 * @throws InvalidKeySpecException
 *
 */
public void ParsMetadata(String xml_metadata)
{
/** parser dell'xml contenente i metadati */
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  DocumentBuilder db;
try
{
db = dbf.newDocumentBuilder();
  InputSource is = new InputSource();
  is.setCharacterStream(new StringReader(xml_metadata));
  Document doc = null;
try
{
doc = db.parse(is);
} catch (SAXException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
    }
```

```
      ......................
  }
}
```

XML

L'Interfaccia verso le API xmht fornite, avviene passando come parametro alle funzioni di publish e get, l'Identificatore dell'oggetto e il Metadato corrispondente sotto forma di stringhe; si è pensato quindi di utilizzare il formato XML [18] per la modellazione dell'Identificatore e del Metadato corrispondente. XML è il formato ideale per lo scambio di informazioni tra sistemi, inoltre è un linguaggio facilmente parsabile, estensibile e auto-descrittivo, indipendente dai linguaggi e piattaforme.

XML (sigla di eXtensible Markup Language) è un metalinguaggio di markup, ovvero un linguaggio marcatore che definisce un meccanismo sintattico che consente di estendere o controllare il significato di altri linguaggi marcatori. Costituisce il tentativo di produrre una versione semplificata di Standard Generalized Markup Language (SGML) che consenta di definire in modo semplice nuovi linguaggi di markup da usare in ambito web. Il nome indica quindi che si tratta di un linguaggio marcatore (markup language) estensibile (eXtensible) in quanto permette di creare tag personalizzati.
Il World Wide Web Consortium (W3C), in seguito alla guerra dei browser (ovvero la situazione verificatasi negli anni novanta nella quale Microsoft e Netscape introducevano, con ogni nuova versione del proprio browser, un'estensione proprietaria all'HTML ufficiale), fu costretto a seguire le individuali estensioni al linguaggio HTML. Il W3C dovette scegliere quali caratteristiche standardizzare e quali lasciare fuori dalle specifiche ufficiali dell'HTML. Fu in questo contesto che

[18] eXtensible Markup Language, http://www.w3.org/XML/

iniziò a delinearsi la necessità di un linguaggio di markup che desse maggiore libertà nella definizione dei tag, pur rimanendo in uno standard. Il progetto XML, che ebbe inizio alla fine degli anni novanta nell'ambito della SGML Activity del W3C, suscitò un così forte interesse che la W3C creò un gruppo di lavoro, chiamato XML Working Group, composto da esperti mondiali delle tecnologie SGML, ed una commissione, XML Editorial Review Board, deputata alla redazione delle specifiche del progetto. Nel febbraio del 1998 le specifiche divennero una raccomandazione ufficiale con il nome di Extensible Mark-up Language, versione 1.0. Ben presto ci si accorse che XML non era solo limitato al contesto web, ma era qualcosa di più: uno strumento che permetteva di essere utilizzato nei più diversi contesti, dalla definizione della struttura di documenti, allo scambio delle informazioni tra sistemi diversi, dalla rappresentazione di immagini alla definizione di formati di dati. Rispetto all'HTML, l'XML ha uno scopo ben diverso: mentre il primo definisce una grammatica per la descrizione e la formattazione di pagine web e, più in generale, di ipertesti, il secondo è un metalinguaggio utilizzato per creare nuovi linguaggi, atti a descrivere documenti strutturati. Mentre l'HTML ha un insieme ben definito e ristretto di tag, con l'XML è invece possibile definirne di propri a seconda delle esigenze.
Di seguito si descrivono i tag definiti per le strutture dati XML relativi all'Identificatore dell'Information Object e dell'owner rispettivamente e infine la struttura XML del Metadato.

```
<identificator>
  <type>
    <typeIO> static/dynamic/anonymous </typeIO>
    <typeHashA> MD5/SHA-1 </typeHashA>
    <typeHashL> MD5/SHA-1 </typeHashL>
```

```
  </type>
  <autenticator> hash(PK_IO) </autenticator>
  <tagL> hash(sCertData) </tagL>
</identificator>

<identificatorO>
  <type>
    <typeO> static/dynamic/anonymous </typeO>
    <typeHashA> MD5/SHA-1 </typeHashA>
    <typeHashL> MD5/SHA-1 </typeHashL>
  </type>
  <autenticator> hash(PK_O) </autenticator>
  <tagL> hash(sCertData) </tagL>
</identificatorO>
```

Anche il proprietario di un Information Object ha un nome, della stessa struttura del nome dell'IO, per cui anche *owner* avrà lo stesso tipo di xml dell'oggetto.

```
<metadata>
  <locator>
      <ipv4> IP address </ipv4>
      <protocol> tcp/udp/... </protocol>
      <port> 1234 </port>
  </locator>
  <attribute>
      <gps> latitude,longitude </gps>
      <semantic> semantic description </semantic>
      <identificatorO> owner's identificator </identificatorO>
  </attribute>
```

```
  <security>
      <sign> sign sCertData with SK-IO </sign>
      <sign_O> sign sCertData with SK-O </sign_O>
      <pk> IO's public key </pk>
      <pk_O> owner's public key </pk_O>
      <typeHash> MD5/SHA-1 </typeHash>
      <typeDSAlg> DSA/RSA </typeDSAlg>
  </security>
</metadata>
```

3.2.4 Proprietà da soddisfare

Autocertificazione

L'Autocertificazione assicura che il dato auto-certificato, chiamato *sCertData*, in un IO è autentico, cioè, nella sua forma originale come generato dal legittimo proprietario. Ricordiamo che sCertData contiene il contenuto del dato e/o metadato di un IO che deve essere autenticato. L'Autocertificazione presuppone che l'ID sia corretto, e autentico. Di conseguenza, l'autocertificazione fornisce una *condizionale integrità del dato*. L'unico modo possibile di effettuare modifiche non autorizzate in sCertData è, cambiando il suo Identificatore.
L'Autocertificazione di IO statici, il cui contenuto non cambia nel tempo, viene garantita includendo il valore hash di sCertData nel campo L dell'Identificatore. La verifica della sCertData viene poi effettuata calcolando il valore hash del dato recuperato, nel nostro caso il metadato e confrontandolo con quello presente nel campo L dell' ID. Se la funzione hash utilizzata è sicura, allora l'unico modo possibile di cambiare sCertData è cambiando l'Identificatore dell'Information Object.

Per gli IO dinamici, in cui i dati contenuti cambiano nel tempo, il valore hash della variabile sCertData non può essere inclusa nell'Identificatore in quanto violerebbe la proprietà della persistenza del nome. In questo caso, l'auto-certificazione si ottiene con la memorizzazione dell'hash di sCertData firmata e autenticata, nel campo *sign_sCertData* dei metadati associati. Più precisamente, si esegue l'hash di sCertData con la funzione hash h, presente nel campo *typeHash* dei metadati di sicurezza e successivamente si firma con la chiave privata SK corrispondente alla specifica chiave pubblica PK nel campo A dell'ID con l'algoritmo DSAlg. Questa firma viene quindi memorizzata nei metadati associati. Più in generale, invece di SK, può anche essere utilizzata un'altra chiave privata autorizzata da SK, dove l'autorizzazione può essere eseguita tramite certificati a chiave pubblica o catene di certificati. SK'/PK' indicano la coppia di chiavi pubblica e privata utilizzata per la firma e per la verifica. Di conseguenza, solo i legittimi proprietari, cioè le entità che conoscono la chiave SK', sono in grado di produrre la firma valida e qualsiasi modifica a sCertData eseguita da soggetti non a conoscenza di SK', quindi non autorizzati saranno rilevati. Tuttavia, se la chiave SK' viene cambiata da un soggetto non autorizzato e viene prodotta una nuova firma valida per la modifica di sCertData, allora l'Identificatore di IO cambierà e, quindi la proprietà di autocertificazione si continua a detenere, perchè è richiesto che l'Identificatore sia autentico. Naturalmente, il legittimo proprietaro può cambiare s-certData e può produrre una nuova firma valida.

Persistenza del nome

NetInf assicura la persistenza dei nomi indipendentemente dai cambiamenti di locazione, cambiamenti del contenuto e del proprietario di un determinato dato/metadato.

L'Indipendenza dalla posizione del dato, è un risultato diretto della divisione del Locatore/Identificatore su cui si basa il framework. I nostri Identificatori sono dinamicamente associati a uno o più specifici locatori, in cui sono memorizzate le copie concrete di IO. Quindi, quando cambia un locatore, l'Identificatore rimane persistente e viene adattato soltanto il suo riferimento con il locatore.

L'Indipendenza dai contenuti soprattutto per i contenuti dinamici è stata realizzata in modo da escludere qualsiasi relazione esplicita di un Identificatore con il contenuto stesso. Questo è il motivo per cui l'hash della firma autocertificata è salvata nei metadati associati e non fa parte dell'Identificatore. Per quanto riguarda i contenuti statici, questa separazione non è richiesta, e l'indipendenza dei contenuti è irrilevante, quindi l'hash del contenuto è parte dell'Identificatore. Pertanto, la flessibilità di NetInf permette un trattamento più semplice per i contenuti statici e assicura la persistenza dei nomi per i contenuti dinamici. Per distinguere gli ID dei contenuti dinamici da quelli dei contenuti statici, viene utilizzato infatti il campo type dell'Identificatore.

L'indipendenza del proprietario sarebbe impossibile da realizzare se, come in DONA, la chiave pubblica PK nel campo A di un ID viene associata ad un proprietario. In NetInf, l'indipendenza del proprietario viene realizzata in due modi. Nell'approccio basico la coppia di chiavi PK/SK è saldamente passata dal precedente proprietario al nuovo proprietario, mentre nel metodo avanzato, il vecchio proprietario autorizza una nuova coppia di chiavi PK'/SK' da essere utilizzata dal nuovo proprietario tramite una chiave pubblica certificata. In entrambi gli approcci, un precedente proprietario di un IO deve essere attendibile per autorizzare un nuovo proprietario, per cui occorre una chiave pubblica certificata da qualche Certification Autority.

A livello implementativo questa proprietà viene garantita con la sola

struttura dati descritta sopra, per cui non è stato necessario nessun metodo ulteriore.

3.2.5 Autenticazione del proprietario

Una peculiarità di NetInf è che l'autenticazione del proprietario è separata dalla auto-certificazione del dato. La coppia di chiavi PK/SK usata per l'autenticazione del proprietario non necessariamente deve essere la stessa di quella utilizzata per l' autocertificazione del dato, ma in alcuni casi speciali possono essere la stessa. A tal fine, ad ogni proprietario, viene assegnato un nome, che può avere la stessa struttura degli Information Object.

$$\text{TYPE}_O \mid A_O = HASH(PK_O) \mid L_O$$

L'etichetta L_O può essere globalmente unica, ad esempio, può includere un nome di una persona o di una compagnia, il suo indirizzo, numero di telefono o altri tag identificativi unici o un insieme di tag di identificazione. I Tag identificativi possono o non possono essere verificabili fisicamente. Per esempio, i tag descritti sopra sono fisicamente verificabili, mentre pseudonimi astratti non lo sono.
Il campo autenticatore $A_O = hash(PK_O)$
è fisicamente verificabile dimostrando la conoscenza della corrispondente chiave segreta SK.
Si effettuano due tipi di autenticazione, la prima più debole chiamata autenticazione e la seconda più forte chiamata Identificazione del proprietario.

L'Autenticazione del prorietario lega essenzialmente il contenuto autocertificato di un IO, il quale include il campo Identificatore dell'oggetto, al campo Autenticator del proprietario. Questo viene ottenuto

includendo A_O in sCertData e firmandolo con entrambi le chiavi SK_{IO} e SK_O, la seconda firma non viene ripetuta se $SK_O = SK'_{IO}$. Le due firme sono inclusi nei metadati di sicurezza. L' Autenticazione del proprietario viene quindi eseguita verificando le due firme.

L'Identificazione del proprietario unisce essenzialmente il contenuto autocertificato di un IO, incluso il suo Identificatore, all' Identificatore del proprietario, che contiene sia il campo A che il campo L del proprietario rispettivamente. Questo viene raggiunto includendo l'Identficatore del proprietario in sCertData e firmandolo con SK_{IO} [19] e con SK_O, dove la seconda firma non viene ripetuta se il $SK_O = SK'_{IO}$. Le due firme sono incluse nei metadati di sicurezza.
Nell'implementazione della tesi si è scelto di soddisfare l'identificazione del proprietario. L'identificazione è un tipo di autenticazione più forte includendo l'intero Identificatore nei metadati. In questo modo è più difficile per un attaccante modificare il campo Type o il campo L essendo inclusi anche loro nel valore hash dei metadati di sicurezza.

L'autenticazione del proprietario e l'identificazione come descritta sopra, si estendono naturalmente anche al caso di più proprietari, inclusi i loro Identificativi in sCertData e fornendo tutte le firme nei metadati di sicurezza. Inoltre Si estende anche agli editori, che sono esclusivamente autorizzati a registrare/cancellare e pubblicare oggetti informativi. I proprietari e gli editori sono autorizzati dai certificati emessi da qualche CA.

[19] o qualsiasi altro autorizzato SK'_{IO}

3.2.6 Implementazione

La classe Lems.java si occupa di effettuare il retrieval, la pubblicazione e la rimozione di un Information Object. E' composta dalle seguenti funzioni principali:

- **public void init (..)**
- **public int publish (String Id, String Metadata)**
- **public int unpublish (String Id, String Metadata)**
- **public int get (String Id, String Metadata)**

init

la funzione init ha il compito di interfacciarsi con la init delle *APIXmht*. Attraverso la funzione init si permette l'apertura di una socket con il nodo *ancora/forwarder* della rete overlay. I parametri della funzione init sono quindi: l'indirizzo Ip, il tipo di protocollo e la porta su cui si deve aprire il socket. In questo modo le comunicazioni tra la libreria passano attraverso le chiamate delle API che comunicano con i nodi dell'overlay sottostante attraverso *socket*.

```
/** importo le API xmht e le sue strutture dati */
import xmhtapi.datastructures.Metadata;
import xmhtapi.XmhtAPI;

public class LeMs {

/** crea l'oggetto XmhtAPI */
XmhtAPI communicationApi = new XmhtAPI();

  public LeMs(){}
```

```
public void init(String overlayNodeIpAddress,
 Integer overlayNodePort,
 int overlayNodeProtocolType)
{
   communicationApi.init(overlayNodeIpAddress,
       overlayNodePort,
       overlayNodeProtocolType);
}
..........
```

Ognuna di queste funzioni deve soddisfare le proprietà di *autocertificazione, persistenza del nome e autenticazione/identificazione del proprietario.*

$$\textbf{publish} = \begin{cases} \text{autocertificazione,} \\ \text{persistenza del nome,} \\ \text{autenticazione del nome} \end{cases} \quad \text{se soddisfatte publishFromName}$$

$$\textbf{unpublish} = \begin{cases} \text{autocertificazione,} \\ \text{persistenza del nome,} \\ \text{autenticazione del nome} \end{cases} \quad \text{se soddisfatte unpublishFromName}$$

La funzione get invece esegue prima la *getFromName*, analizza la lista dei metadati ricevuti e se soddisfano le proprietà crittografiche vengono ritornati all'applicazione chiamante, altrimenti la get fallisce.

publish

La funzione di publish permette la pubblicazione di un Information Object, che equivale ad una funzione di *put* nelle *hash table distribuite*, dove la chiave è data dall'*identificatore*.

```
/**
* @throws TransformerException
* @throws IOException
* @throws NoSuchAlgorithmException
* @throws InvalidKeySpecException
* @throws InvalidKeyException
* @throws SignatureException
*
*/
public int publish(String Identificatore,String Metadata)
{
/** parsing xml dell'identificaatore e del metadato */
parser(Identificatore,Metadata);

  /** effettua controlli di sicurezza */
if(security(identificator,metadataS))
  {
/** se i controlli sono andati a buon
  * fine chiama la publish di xmhtapi */
return communicationApi.publishFromName
(Identificatore, Metadata,false);
}
return -1;
}
```

unpublish

La funzione di unpublish permette la rimozione di un Information Object dalla rete [20].

```
/**
* @throws TransformerException
* @throws IOException
* @throws NoSuchAlgorithmException
* @throws InvalidKeySpecException
* @throws InvalidKeyException
* @throws SignatureException
*
*/
public int unpublish(String Identificatore,String Metadata)
{
/** parsing xml dell'identificaatore e del metadato */
parser(Identificatore,Metadata);

  /** effettua controlli di sicurezza */
  if(security(identificator,metadataS))
  {
/** se i controlli sono andati a buon
  * fine chiama unpublish di xmhtapi */
return communicationApi.unpublishFromName
(Identificatore, Metadata,false);
}
return -1;
}
```

[20] da non confondere con la cancellazione dell'Information Object

get

La funzione get, permette il recupero dei metadati associati all'identificatore richiesto. In questo caso la funzione getFromName delle APIxmht viene chiamata prima di effettuare i controlli di sicurezza, in quanto prima di tale chiamata non si ha a disposizione i metadati da confrontare. I metadati che soddisfano le proprietà di sicurezza attraverso la funzione security, vengono restituiti come valore di ritorno della funzione get.

```
/**
* @throws TransformerException
* @throws IOException
* @throws NoSuchAlgorithmException
* @throws InvalidKeySpecException
* @throws InvalidKeyException
* @throws SignatureException
*
*/
public int get(String Identificatore,Vector<Metadata> res)
{
Vector<Metadata> metaReceive = new Vector<Metadata>();
String resourceName = new String();
  resourceName = Identificatore;

try {
communicationApi.getFromName(resourceName, metaReceive);
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
```

```
  /** parsing xml identificatore */
  identificator.ParsIdentificator(Identificatore);

/** scorro i metadati */
for(int i = 0; i < metaReceive.size(); ++i)
{
 /** parsing xml del metadato */
 metadataS.ParsMetadata(metaReceive.elementAt(i).getMetadata());

 /** effettuo i controlli di sicurezza su ogni metadato
 * della lista dei metadati ricevuti */
if(selfCertification(identificator,metadataS))

  /** il metadato che soddisfa i controlli sopra viene
* viene aggiunto alla lista res contenente i metadati richiesti */
  res.add(metaReceive.elementAt(i));
}
return 0;
}
```

Parser

Sono forniti più modi per effettuare il parsing di un documento XML. Un parser è un programma che effettua la lettura di un documento XML e lo divide in blocchi discreti. I due classici approcci per processare i documenti XML sono:

- Simple API for XML Processing (SAX)
- Document Object Model (DOM)[24]

SAX è un'API di basso livello il cui principale punto di forza è l'efficienza. Quando un documento viene parsato usando SAX, una serie di eventi vengono generati e passati all'applicazione tramite l'utilizzo di callback handlers che implementano l'handler delle API SAX. Gli eventi generati sono di livello molto basso e devono essere gestiti dallo sviluppatore che, inoltre, deve mantenere le informazioni necessarie durante il processo di parsing. Oltre ad un utilizzo piuttosto complicato, SAX soffre di due limitazioni di rilievo: non può modificare il documento che sta elaborando e può procedere alla lettura solo *in avanti*: non può tornare indietro. Quindi, quello che è stato letto è perso e non è possibile recuperarlo. DOM, invece, ha come punto di forza la semplicità d'utilizzo. Una volta ricevuto il documento, il parser si occupa di costruire un albero di oggetti che rappresentano il contenuto e l'organizzazione dei dati contenuti. In questo caso l'albero esiste in memoria e l'applicazione può attraversarlo e modificarlo in ogni suo punto. Ovviamente il prezzo da pagare è il costo di computazione iniziale per la costruzione dell'albero ed il costo di memoria [25]. Oltre ad effettuare il parsing estraendo i valori xml, alcune volte è anche necessario modificare l'xml stesso, per cui è stato scelto il DOM.

```
/**
* @throws TransformerException
* @throws IOException
* @throws NoSuchAlgorithmException
* @throws InvalidKeySpecException
*
*/
public void parser(String Identificatore,String Metadata)
{
/** parsing xml identificatore */
```

```
identificator.ParsIdentificator(Identificatore);

/** parsing xml metadata */
  metadataS.ParsMetadata(Metadata);
}
```

Chiave pubblica e privata

Le chiavi pubbliche dell'Information Object e del relativo owner rispettivamente introdotte nei metadati di sicurezza si assumono siano create di tipo PuclicKey e PrivateKey utilizzando *java.security.KeyPair*. Questa chiave deve essere convertita in base 64 per poterla scrivere adeguatamente in xml. Viene convertita utilizzando:

java.lang.Object
javax.xml.bind.DatatypeConverter

Vengono utilizzate le seguenti funzioni statiche per scrivere e leggere la chiave pubblica.

```
/** scrive la chiave pubblica in xml
  * convertendo da array byte in Stringa*/
static java.lang.String printBase64Binary (byte [] val)

/** legge la chiave pubblica da xml
  * converte da Stringa in array di byte /
static byte [] parseBase64Binary
(lexicalXSDBase64Binary java.lang.String)
```

Funzioni crittografiche

Vengono utilizzate delle funzioni crittografiche per l'hashing della chiave pubblica e/o del contenuto del metadato nel caso dell'Information Object statico utilizzando la funzione hash richiesta; la seconda funzione crittografica consiste nella firma digitale del dato/metadato utilizzando l'algoritmo presente nell'xml del metadato e infine le rispettive funzioni per la verifica del valore hash e della firma digitale.

Funzioni hash one-way

Una *funzione hash f:* $X \to Y$ è definita per un dominio X e un codominio Y finiti e tale che $n = |X| >> m = |Y|$. Nella costruzione di algoritmi queste funzioni sono impiegate per operare su elementi $x \in X$ attraverso la loro *immagine (o fingerprint)* $y = f(x) \in Y$, essenzialmente per due scopi. Anzitutto la rappresentazione di y richiede $\log_2 m$ bit ed è quindi più breve di quella di x che ne richiede $\log_2 n$; inoltre Y può avere una struttura assente in X, per esempio gli elementi y corrispondono alle posizioni di un vettore di m celle in cui memorizzare informazioni associate a m diversi elementi di X.
La differenza di cardinalità tra i due insiemi X e Y implica che esiste una partizione di X in sottoinsiemi disgiunti $X_1, \ldots, X_m$ tali che, per ogni valore dell'indice i, tutti gli elementi in X_i hanno come immagine uno stesso elemento di Y.
Le funzioni hash applicate in crittografia inoltre devono soddisfare ulteriori proprietà, in particolare rivestono una certa importanza le *funzioni hash one-way*, tali che:

1. Per ogni elemento $x \in X$ è computazionalmente facile calcolare *f(x)*.
2. Per la maggior parte degli elementi $y \in Y$ è computazionalmente difficile determinare un x tale che f(x) = y.

3. E' computazionalmente difficile determinare una coppia di elementi x_1, x_2 in X tali che $f(x_1) = f(x_2)$.

Tutte le funzioni hash dovrebbero soddisfare la prima proprietà. La seconda (proprietà one-way) e la terza (proprietà di *claw-free*, opzionale), necessarie per impieghi crittografici, dovrebbero essere garantite attraverso una dimostrazione formale di intrattabilità: poichè però le funzioni così costruite sono difficili da utilizzare, ci si accontenta di funzioni semplici per cui siano falliti in pratica tutti i tentativi di attacco. In questa condizione si trovano le due funzioni hash one way più usate in crittografia, dette *MD5* e *SHA* [7][p.131].
la funzione MD5 (*Message Digest* versione 5 proposta da Rivest nel 1992) è utilizzata molto frequentemente per controllare l'integrità dei messaggi su linee sicure, benchè non soddisfi in pieno la proprietà di claw-free. Se questa proprietà è cruciale per la sicurezza del sistema si può adottare la SHA (*Secure Hash Algorithm*), progettata da NIST e NSA nel 1992. la SHA è una funzione crittograficamente sicura nel senso che soddisfa tutti i requisiti richiesti per le funzioni hash one-way e inoltre genera immagini molto diverse per sequenze molto simili. Con il termine SHA si indica una famiglia di cinque diverse funzioni crittografiche di hash, dette rispettivamente SHA-1, SHA-224, SHA-256, SHA-384 e SHA-512. Il primo produce un digest del messaggio di soli 160 bit, mentre gli altri producono digest di lunghezza in bit pari al numero indicato nella loro sigla (SHA-256 produce un digest di 256 bit). L'SHA-1 è il più diffuso algoritmo della famiglia SHA ed è utilizzato in numerose applicazioni e protocolli.
LEMS permette l'utilizzo delle due funzioni crittografiche MD5 e SHA-1. Il parametro *typeHash* definisce il tipo di funzione da utilizzare. Da notare che la funzione passata come parametro deve essere la stessa utilizzata per il campo autenticator nel caso di verifica della chiave

pubblica, oppure uagule alla funzione utilizzata per il valore hash del metadato contenuto nel campo L nel caso di IO statico.
questa funzione esegue l'hash della stringa *str* utilizzando la funzione hash *typeHash* passati come parametro, e restituisce un array di byte.

```
public byte[] hash(String str, String typeHash){
    MessageDigest sha = MessageDigest.getInstance(typeHash);
    sha.update(str.getBytes());
    return sha.digest();
}

public boolean checkHash(byte[] impronta, byte[] hash){
    if (MessageDigest.isEqual(impronta,hash))
        return true;
    else
        return false;
}
```

Frima digitale

Nei protocolli crittografici sono richieste tre funzionalità importanti a seconda dell'uso del livello di protezione desiderato. La firma digitale è la funzionalità più complessa e include le altre due. Esse sono:

Identificazione Un sistema di elaborazione isolato o inserito in una rete deve essere in grado di accertare l'identità di un utente che richiede di accedere ai suoi servizi.
Autenticazione Il destinatario di un messaaggio deve essere in gra-

do si accertare l'identità del mittente e l'*integrità* del crittogramma ricevuto, cioè che questo non sia stato modificato o sostituito nella trasmissione. Deve quindi essere difficile a un intruso spacciarsi per un altro utente o modificare i messaggi da questo inviati.

Firma digitale risulta neccessaria se il mittente e il destinatario non sono tenuti a fidarsi l'uno dell'altro. La firma digitale deve soddisfare tre requisiti: (1) il mittente non deve poter negare di aver inviato un messaggio m; (2) il destinatario deve poter accertare l'identità del mittente e l'iintegrità del crittogramma ricevuto (cioè deve poter autenticare il messaggio); (3) il destinatario non deve poter sostenere che un messaggio $m' \neq m$ è quello inviatogli dal mittente. Questi requisiti devono inoltre essere verificabili da una terza parte, cioè da un giudice possibilmente chiamato a certificare il corretto svolgimento della comunicazione.[21].[7][p.136]

Per progettare firme digitali si possono impiegare sia i cifrari simmetrici che quelli asimmetrici, i primi danno luogo a realizzazioni più complicate e computazionalmente costose, per cui LEMS utilizza il classico meccanismo proposto da Diffie e Hellman (DH) basato sui cifrari asimmetrici, in cui la chiave privata del mittente (nel nostro caso dell'Information Object e/o del proprietario dell'IO) è utilizzata da questi per produrre la firma, e la sua chiave pubblica è utilizzata dalla libreria per verificarne l'autenticità.

La piattaforma Java (Java2) mette a disposizione, attraverso il framework Java Cryptography Architecture (JCA), una serie di facility per il supporto della firma digitale e dei servizi di crittografia più comunemente utilizzati.

Si assume che la chiave pubblica venga distribuita attraverso un certificato reso disponibile da un ente certificatore.

[21] vedere più avanti le CA

Gli algoritmi utilizzati per la firma digitale sono *Digital Signature Algortithm (DSA)* e *(RSA)*. Questi algoritmi solitamente non vengono applicati direttamente al messaggio (di lunghezza variabile), ma su un *impronta o MessageDigest* (di lunghezza fissa), ricavata da quest'ultimo, la cui univocità è garantita dall'algoritmo di Hashing utilizzato per ottenerla.

JCA è un framework estendibile ed interoperabile attraverso il quale in Java 2 viene reso disponibile un insieme completo di servizi di crittografia. Inoltre vengono fornite le API per il supporto della firma digitale.

In JCA uno dei concetti più interessanti è quello del Provider : un vendor di servizi crittografici resi disponibili e manipolabili attraverso il framework stesso. Per aggiungere un Provider al framework è necessario estendere la classe java.security.Provider.

In questa tesi si è utilizzato il provider *Sun*, di default, che il JCA ci mette a disposizione.

```
public byte[] DSAlg(PrivateKey, byte[] data, String typeDSAlg){
    Signature dsa = Signature.getInstance(typeDSAlg);
    dsa.initSign(priv);
    dsa.update(data);
    return dsa.sign();
}
```

effettua la firma digitale sul dato sCertData usando l'algoritmo richiesto

```
public boolean verDSAlg(PublicKey pub, byte[] data, byte[]
firma, String typeDSAlg){
    Signature dsa = Signature.getInstance(typeDSAlg)
```

```
    dsa.initVerify(pub);
    dsa.update(data)
    return dsa.verify(firma);
}
```

Per un ulteriore cifratura del dato, si può estendere il package JCA con *Java Cryptography Extension (JCE).*[16]

security

Questa funzione si occupa di effettuare tutti i controlli di sicurezza per soddisfare le proprietà di *autocertificazione, persistenza del nome e autenticazione del proprietario* viste sopra.

```
/**
* @throws SignatureException
* @throws InvalidKeyException
* @throws NoSuchAlgorithmException
* @throws InvalidKeySpecException
*
*/
public boolean security(IdentifierS identificatore,
                        MetadataS metadata)
{
  /** se Information Object dinamico */
if(identificatore.typeIO_.compareTo("dynamic")==0)
 /** controlli per IO dinamici */
 return securityDynamic(identificatore,metadata);
else
 /** controlli per IO statici */
```

```
 return securityStatic(identificatore,metadata);
}

/**
  * @throws NoSuchAlgorithmException
*
*/
public boolean securityStatic(IdentifierS identificatore,
                              MetadataS metadata)
{
/** variabile di controllo */
boolean bRet = false;

/** creo il valore hash del
  * metadato ricevuto utilizzando
* utilizzando la funzione
* hash definita in typeHashL*/
byte[] hash_metadata =
hash(metadata.metadata_.getBytes(),identificatore.typeHashL_);
  byte[] L =
  DatatypeConverter.parseBase64Binary(identificatore.tagL_);

/** verifica dei due valori hash,
  * in modo da garantire l'integrita'
* del metadato confrontandolo con
* il campo L dell'Identificatore */
if (CheckHash(hash_metadata,L))
  bRet = true;

return bRet;
```

```
}

/**
* @throws SignatureException
* @throws InvalidKeyException
* @throws NoSuchAlgorithmException
* @throws InvalidKeySpecException
*
*/
public boolean securityDynamic(IdentifierS identificatore,
                               MetadataS metadata)
{
  /** variabili di controllo */
  boolean verified  = false;
  boolean verifiedO = false;

  /** recupero la chiave dai metadati di
    * sicurezza e la
    * converto per effetuare l'hash, in
    * modo da fare poi
    * la verifica con l'hash della chiave
    * pubblica presente
    * nel campo A dell'Identificatore */
  byte[] key_m =
  DatatypeConverter.parseBase64Binary(metadata.pk_);
  byte[] improntaPk = hash(key_m, metadata.typeHash_);

  /** recupero l'hash della chiave pubblica presente
  * nel campo A dell'Identificatore */
  byte[] hash_A =
```

```
  DatatypeConverter.parseBase64Binary
  (identificatore.autenticator_);

  /** verifica dei due valori hash,
    * in modo da garantire
    * l'integrita' della chiave
    * pubblica presente
    * nell'Identificatore e nel
    * metadato di sicurezza */
if (CheckHash(improntaPk,hash_A))
{
  /** converto la chiave pubblica
    * di IO da byte[] in PublicKey
   * utile per il confronto della
   * firma digitale e garantire
  * l'autocertificazione */
    X509EncodedKeySpec pubKeySpec =
    new X509EncodedKeySpec(key_m);
    KeyFactory keyFactory =
    KeyFactory.getInstance("DSA");
    PublicKey pubKey =
    (PublicKey)keyFactory.generatePublic(pubKeySpec);

    /** coverto la stringa del
      * metadato ricevuta in byte
      * per poter confrontare la
      * firma digitale presente
      * nei metadati associativi */
    byte[] metaConv =
    DatatypeConverter.parseBase64Binary
```

```
  (metadata.metadata_);

  /** converto la stringa della firma
    * con SK_IO in byte per il confronto */
  byte[] f =
  DatatypeConverter.parseBase64Binary(metadata.sign_);

  /** identificazione del proprietario
    * verificando la firma
    * di identificatorO nei metadati */
  byte[] key_mO =
  DatatypeConverter.parseBase64Binary(metadata.pk_O_);

  /** converto la chiave pubblica
    * di Owner da byte[] in PublicKey
* utile per il confronto della
* firma digitale e garantire
* l'autenticazione del proprietario */
  X509EncodedKeySpec pubKeySpecO =
  new X509EncodedKeySpec(key_mO);
  KeyFactory keyFactoryO =
  KeyFactory.getInstance("DSA");
  PublicKey pubKeyO =
  (PublicKey)keyFactoryO.generatePublic(pubKeySpecO);

  /** converto la stringa della
    * firma con SK_O in byte
    * per il confronto */
  byte[] fO =
  DatatypeConverter.parseBase64Binary(metadata.sign_O_);
```

```
try
  {
  /** il framework Java JCA mette
    * a disposizione delle API
    * per il supporto della firma digitale,
    * applicando direttamente la
    * funzione hash prima della verifica,
    * in questo modo mi creo la stringa completa
    * (funzione hash e algoritmo di firma)
    * da passare come parametro alla
    * funzione VerDSAlg */
  String DSAlg = new String();
  if(metadata.typeHash_.compareTo("SHA-1")==0)
  {
     DSAlg = "SHA1with"+metadata.typeDSAlg_;
  }
  else
  {
    DSAlg = metadata.typeHash_+"with"+metadata.typeDSAlg_;
  }

  /** verifico la firma digitale effettuata
    * con la chiave privata SK_IO
    * del metadato con la chiave pubblica
    * dell'Information Object */
  verified = VerDSAlg(pubKey,metaConv,f,DSAlg);

  /** verifico la firma digitale
    * effettuata con la chiave privata SK_O
```

```
        * del metadato con la chiave pubblica
        * dell'owner di IO */
      verifiedO = VerDSAlg(pubKeyO,metaConv,fO,DSAlg);
    }
    catch (SignatureException se)
    {
      System.out.println("La firma con la chiave SK ha un
                          formato non valido!");
    }
  }

  /** verifico l'esito delle due firme */
if(verified && verifiedO)
return true;
else
return false;
}
```

Capitolo 4

Conclusioni

4.1 Test e Conclusioni

Il protocollo realizzato è stato testato e analizzato principalmente con l 'uso del debug di eclipse. Sono stati analizzati, bit per bit i messaggi cifrati e serializzati prima dell'invio, e i relativi messaggi ricevuti, deserializzati e decifrati.
Il protocollo di autenticazione viene utilizzato nel momento della Join da parte di un generico nodo, da quel momento in poi il nodo può pubblicare qualsiasi risorsa senza nessun tipo di autenticazione aggiuntiva. La valutazione del protocollo di autenticazione non influisce particolarmente sulle prestazioni della rete, sebbene ci sia un overhead relativo alle funzioni di cifratura e decifrazione. Il protocollo sopra descritto garantisce l'affidabilità dei nodi che fanno parte dell'overlay, ma non garantisce l'affidabilità e l'integrità delle risorse pubblicate, in quanto qualsiasi entità può sfruttare il nodo XMHT per pubblicare gli oggetti, senza nessun tipo di controllo. Questo aspetto è stato raggiunto grazie alla libreria LEMS (Library Extensible Metadata Security).

Le diverse proprietà di sicurezza sono basate sul fatto che gli identificatori contengono l'hash della chiave pubblica, tipicamente ma non

necessariamente appartenenti al proprietario. Questa chiave pubblica consente al proprietario di autenticare il contenuto variabile di un IO dinamico, assicurando che tutti i dati e tutti i metadati del'IO selezionato siano autentici se il nome dell' Information Object è autentico. D'altra parte il contenuto di un IO statico viene autenticato inserendo l'hash del contenuto stesso nell'Identificatore dell' IO.

Avendo il contenuto hash della chiave pubblica del proprietario come parte dell'identificatore offre intrinsecamente la possibilita' di autenticare l'owner o il publisher. In molti casi, owner e publisher sono la stessa cosa, altre volte invece sara' necessario autenticare questi diversi attori separatamente (ad esempio nel caso di un brano musicale, la canzone viene creata dall'artista e pubblicata dalla casa discografica). Lems consente l'autenticazione di molti attori (creatori/editori etc) per lo stesso oggetto, includendo i dati aggiuntivi di autenticazione nei metadati. Senza queste informazioni aggiuntive relative all'autenticazione del ruolo, il proprietario rimane anonimo.

Inoltre in un qualsiasi momento un proprietario puo' cambiare la sua chiave pubblica e diventare anonimo. In piu' se il proprietario cambia, il nuovo proprietario dello stesso IO puo' essere autenticato. E' interessante notare che in tutte queste operazioni, il nome dell' Information-Object rimane lo stesso, in quanto la chiave pubblica del proprietario non e' legata in nessun modo al nome dell'oggetto ma soltanto ai suoi metadati. Anche quest'ultima proprietà, *persistenza del nome* è stata raggiunta.

La libreria Lems insieme al protocollo di autenticazione aumentano il grado di sicurezza dell'overlay xmht, che in questo modo può essere utilizzato per lo sviluppo di applicazioni che necessitano la *confidenzialità e l'integrità* delle informazioni.

4.1.1 Sviluppi Futuri

La libreria crittografica Lems favorisce l'utilizzo dell'overlay xmht anche ad applicazioni che necessitano uno scambio di informazioni integro e confidenziale. Con le proprietà introdotte si ha una implementazione base, un *prototipo* di NetInf, sviluppato in java capace di essere utilizzato per applicazioni create da zero o estendere applicazioni già esistenti (ad esempio, plugin Firefox, Thunderbird).
Nel caso del plugin Firefox per esempio, si potrebbe pensare di interpretare i link che contengono gli Identificatori NetInf invece dell'URL. Questo porterebbe alcuni vantaggi all'utente:

- controllo automatico dell'integrità dei dati
- riduzione dei collegamenti interrotti

Inoltre anche gli Editori, *i publisher* avrebbero dei vantaggi:

- semplificazione della gestione dei contenuti, avendo un identificatore persistente

Inoltre si potrebbe pensare di integrare facilmente una versione leggera per cellulari Android senza problemi.

Un sviluppo futuro più interessante potrebbe essere anche l'utilizzo della libreria per applicazioni sanitarie, nell'ottica di supportare soluzioni innovative basate sulle tecnologie dell'informazione e della comunicazione, volte a migliorare le condizioni e la qualità di vita delle persone anziane (AAL). [1] [26]

[1] Ambient Assisted Living, programma comunitario congiunto istituito attraverso l'articolo 169 del Trattato dell'Unione Europea come unione di diversi programmi di ricerca nazionali, al fine di supportare progetti per lo sviluppo di soluzioni innovative basate sulle tecnologie dell'informazione e della comunicazione volte a migliorare le condizioni e la qualità di vita delle persone anziane.

Lo sviluppo tecnologico, non solo delle apparecchiature elettromedicali e dei farmaci, ma anche dei servizi, ha contribuito e contribuirà sempre di più al miglioramento degli standard di salute e di qualità della vita dei cittadini e all'evoluzione di specifici servizi, quali ad esempio la telemedicina e la cartella clinica elettronica. Questi servizi, che di fatto facilitano l'accesso alle prestazioni sanitarie, sono però ancora poco diffusi, non interconnessi fra loro e lasciano quindi ampio margine di miglioramento.[19] Ad esempio creare un sistema per il monitoraggio dei parametri ambientali e umani che consenta la definizione di un modello comportamentale dei pazienti, necessario all'identificazione di situazioni anomale o non attese dal loro stile di vita, sulla base delle quali generare allarmi che attivino l'intervento da parte di terzi (operatori sociali, parenti, etc).[19] Un ruolo fondamentale per garantire l'integrità nella trasmissione del dato, la sicurezza e la qualità del servizio (QoS). La tematica della gestione della rete con il moltiplicarsi degli oggetti connessi include non solo problematiche di disponibilità di indirizzo, ma anche quelle di volume di dati. Negli Stati Uniti sono già stati siglati accordi interoperatore per la creazione di reti dedicate all'IoT e all'M2M. [2][19] [27]

SCENARI SORPRENDENTI: L'Internet delle cose, il Machine 2 Machine non sono quindi che due aspetti dello stesso processo di innovazione, un processo che non solo non si arresterà, ma che riserverà scenari quasi sorprendenti. Qualcuno ha già iniziato addirittura a ipotizzare i paradigmi del Web 4.0, una Rete capace di predire gli eventi. Stiamo a vedere cosa succederà..

[2]Machine to Machine, contribuisce a creare un vero e proprio nuovo modello di informatica basato sulla gestione degli eventi.

Ringraziamenti

Per il sostegno nella realizzazione di questa tesi ringrazio in particolar modo il professore Maurizio Bonuccelli e il professore Roberto Grossi.
Ringrazio soprattutto mio padre e tutta la mia famiglia per avermi sostenuto non solo economicamente ma anche moralmente in tutti questi anni, credendo in me fino alla fine, il merito di questa tesi è soprattutto vostro.
Grazie a tutti gli amici che in questi anni mi hanno sostenuto nei momenti difficili, soprattutto Carmelo, Antonio, Irena, Mimmo, Ernesto etc etc, in particolare Antonella che mi è rimasta vicino fino alla fine.
Grazie a tutti.
Giuseppe

Bibliografia

[1] Federico Andreini, Flavio Crisciani, Claudio Cicconetti, Raffaella Mambrini *Context-Aware Location in the Internet of Things*, Intecs S.p.a

[2] Bengt Ahlgren, Matteo D'Ambrosio, Christian Dannewitz, Marco Marchisio, Ian Marsh, Börje Ohlman, Kostas Pentikousis, René Rembarz, Ove Strandberg, Vinicio Vercellone, *Design Considerations for a Network of Information.*

[3] Bengt Ahlgren, Matteo D'Ambrosio, Christian Dannewitz,Anders Eriksson, Jovan Goliæ, Björn Grönvall, Daniel Horne, Anders Lindgren, Olli Mämmelä, Marco Marchisio, Jukka Mäkelä, Septimiu Nechifor, Börje Ohlman, Kostas Pentikousis, Sabine Randriamasy, Teemu Rautio, Eric Renault, Pasi Seittenranta, Ove Strandberg, Bogdan Tarnauca, Vinicio Vercellone, Djamal Zeghlache *Objective FP7-ICT-2007-1-216041/D-6.2 The Network of the Future Project 216041 4WARD Ű Architecture and Design for the Future Internet D-6.2 Second NetInf architecture description.*

[4] Federico Andreini, Flavio Crisciani, Claudio Cicconetti, Raffaella Mambrini*P2P for ZigBee M2M Services in the Internet of*

Things, Telecommunications Business Unit - Intecs S.p.a.

[5] Christian Dannewitz, Kostas Pentikousis, René Rembarz, Éric Renault, Ove Strandberg, Javier Ubillos *Scenarios and Research Issues for a Network of Information.*

[6] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, Ion Stoica *Towards a Common API for Structured Peer-to-Peer Overlays*, MIT Laboratory for Computer Science, Cambridge, MA. University of California, Berkeley, CA. Rice University, Houston, TX.

[7] Paolo Ferragina, Fabrizio Luccio *Crittografia, Principi Algoritmi Applicazioni.*

[8] V. Jacobson, M. Mosko, D. Smetters, and J. Garcia-Luna-Aceves *Content-centric networking*,Whitepaper, Palo Alto Research Center, January 2007.

[9] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec *The many faces of publish/subscribe*,ACM Computing Surveys, 35(2):114Ű131, 2003.

[10] M.Caesar, T. Condie, J.Kannan, K.Lakshminarayanan, and I. Stoica, ROFL: Routing on Flat Labels, *ACM SIGCOMM Computer Communication Review*, vol. 36, no.4, p. 363, Aug.2006.

[11] S.Pack, K.Park, T. Kwon, and Y. Choi, *SAMP: Scalable Application-Layer Mobility Protocol*, IEEE Communication

Magazine, no. June, pp. 86-92, 2006.

[12] X. Shen, H. Yu, J.Buford, and M.Akon, Eds.*Handbook of Peer-to-Peer Networking*, Springer, 2010.

[13] D. Rowstron *Pastry Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, in IFIP/ACM International Conference on Distributed System Platform (Middleware), 2001.

[14] http://it.wikipedia.org/wiki/Advanced_Encryption_Standard.

[15] http://docs.oracle.com/javase/1.4.2/docs/guide/security/-CryptoSpec.htmlArchitecture.

[16] http://docs.oracle.com/javase/1.4.2/docs/guide/security/-jce/JCERefGuide.html.

[17] http://saperi.forumpa.it/story/63469/internet-delle-cose-guidare-la-ricerca-luniversita-di-pisa.

[18] Euro Beinat *L'Internet delle cose: da target individuali a collective sensing, il new business nel mondo iperconnesso*, http://www.webcopywriter.it/news/linternet-delle-cose-e-qui/, 8a giornata della comunicazione, 8.04.11. Castello di Udine.

[19] http://www.cwi.it/2011/04/20/linternet-delle-cose-e-il-settore-sanitario-modelli-di-business-ed-esempi-di-servizi/4/.

[20] http://www.4ward-project.eu/.

[21] *(RDF)*,http://www.w3.org/RDF/, http://www.w3.org/2001/sw/.

[22] http://www.iet.unipi.it/g.dini/Teaching/ssi/materiale-didattico/Analisi-progetto-cryptoproto.pdf

[23] M.Caiazza, G. Laccetti, G.Schmid *Il Metodo Induttivo e la verifica di un protocollo crittografico*, Consiglio Nazionale delle Ricerche Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR.

[24] http://www.w3.org/DOM/.

[25] http://www.link.it/isi/jsp/index.jsp?sel=doc-fileguida=Dispensarel=1.0id=ar01s03.htmlid2793538.

[26] http://www.aal-europe.eu/

[27] http://www.datamanager.it/rivista/rubriche/dall-internet-delle-cose-al-machine-2-machine

Appendice A

Source Code Applicazione di Test

Si riporta il codice sorgente di una piccola applicazione per testare il funzionamento della libreria Lems.

```
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.util.Vector;
import javax.xml.bind.DatatypeConverter;
import xmhtapi.Global;
import xmhtapi.datastructures.Metadata;
import Lems.IdentifierS;
import Lems.LeMs;

public class ApplicationTest {

/**
 * @param args
 * @throws Exception
 */
public static void main(String[] args)
throws Exception {

/** tempo corrente in millisecondi */
long timeNow = 0;

/** tempo di ritorno metadato */
long timeGet = 0;

/** inizializzo le Axs api e le xmhtapi */
LeMs secureApi = new LeMs();
secureApi.init("192.168.2.6", 4000,
```

```
                Global.SOCKET_TYPES.TCP,10000);

/**creo un identificatore di esempio */
IdentifierS identifier = new IdentifierS();
String metadata = new String();

String apice = "\"";

/** inizializzo la variabile
  * da inserire nel campo A
  * dell'Identificatore */
String autenticator = new String();

/** genero la coppia di chiavi PK/SK,
  * pubblica e privata rispettivamente,
  * appartenente all'Information Object,
  * utile per verificare l'integritàã
  * dell'Identificatore inserendo l'hash
  * di PK nel campo A di ID */
try {
secureApi.ProductKey();
secureApi.ProductKeyO();
} catch (NoSuchAlgorithmException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
} catch (NoSuchProviderException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}

/** costruisco parte del metadato,
  * da completare in maniera diversa
  * in base al tipo dell'oggetto
  * presente nel campo type dell'Identificatore
  * PS: il metadato parziale serve per poter
  * effettuare l'hash o la fimra in modo
  * da creare s-certData */

    //inizialmente vuoti per IO statico
String keyPK     = new String();
String tipoHash  = new String();
String tipoDSAlg = new String();

//vuoti per IO dinamico
String typeHashA = new String();
String typeHashL = new String();

/** firma di s-certData, nel nostro caso
  * la firma del metadato, utile soltanto
```

```
  * nel caso che l'oggetto sia di tipo
  * dinamico, in qeusto caso il metadato
  * viene firmato con SK_IO e inserito
  * nei metadati associativi */
String f_sCertData = new String();

/** firma di s-certData, nel nostro caso la
  * firma del metadato, utile soltanto nel
  * caso che l'oggetto sia di tipo
  * dinamico, in qeusto caso il metadato
  * viene firmato con SK_O e inserito
  * nei metadati associativi */
String f_sCertDataO = new String();

/** costruisco i metadati formattati
  * i xml contenenti tutte le
  * informazioni relative alla sicurezza
  * per l'information Object corrispondente
  * all'identificatore sopra creato */
String metadataParziale = "<?xml version="+apice+"1.0"+apice+
" encoding="+apice+"UTF-8"+apice+
" standalone="+apice+"no"+apice+"?>+
"<metadata><locator>"+
"<ipv4>192.168.27.67</ipv4>"+
"<protocol>tcp</protocol>"+
"<port>4000</port></locator>"+
                        "<attribute><gps>43716164,10396492</gps>"+
                        "<semantic>video</semantic>"+
                        "<identificatorO/></attribute>"+
                        "<security><sign/><sign_O/><pk/>"+
                        "<pk_O/><typeHash/><typeDSAlg/>"+
                        "</security></metadata>";


/**creo il tipo dell'identificatore */
String typeIO = new String("dynamic");
String tagL = new String();

/** nel caso il tipo dell'identificatore
  * sia statico, va fatto l'hash del
  * metadatoParziale e inseriyto
  * nel campo L dell'Identificatore */
if ((typeIO.compareTo("static")==0) ||
    (typeIO.compareTo("anonymous")==0))
{
/** creo l 'identificatore */
typeHashA = new String("SHA-1");
typeHashL = new String("SHA-1");
```

```
byte[] hash_metadata = secureApi.hash
(metadataParziale.getBytes(),
"SHA-1");

/** assegno l'hash del metadato
  * al campo L di ID */
tagL = DatatypeConverter.printBase64Binary(hash_metadata);
metadata = metadataParziale;
}

/** solo nel caso l'Information Object
  * è di tipo dinamico altrimenti se
  * IO Anonymous oppure static
  * l'oggetto viene considerato
  * liberamente pubblicabile,
  * senza la necessitàã di alcuna autenticazione */
else if (typeIO.compareTo("dynamic")==0)
{
String typeHashAO = new String("SHA-1");

//credo sia sempre dinamico
String typeO = new String("dynamic");

String keyPKO = new String();

/** memorizzo la chiave pubblica
  * dell'Information Object nei
  * metadati di sicurezza
  * in modo da garantire l'integrità
  *ãdella chiave verificandola
  * con l'hash presente nell'Identificatore */
keyPK = DatatypeConverter.printBase64Binary
(secureApi.PK_.getEncoded());

/** memorizzo la chiave pubblica
  * dell'owner dell'IO nei metadati
  * di sicurezza in modo da garantire
  * l'integritàãdella chiave verificandola
  * con l'hash presente nell'Identificatore */
keyPKO = DatatypeConverter.printBase64Binary
(secureApi.PK_O.getEncoded());

/** assegno il tipo di funzione
  * hash e il tipo di algortimo utilizzato
      * per la firma digitale dell'oggetto*/
tipoHash  = new String("SHA-1");
tipoDSAlg = new String("DSA");

/** creo l'hash della chiave pubblica
```

```
  * PK dell'Information Object
    * da inserire nel campo A
    * dell'Identificatore */
byte[] improntaHash = secureApi.hash
  (secureApi.PK_.getEncoded(),"SHA-1");

  /** assegno l'hash della chiave
    * pubblica al campo A di ID */
  autenticator = DatatypeConverter.printBase64Binary
  (improntaHash);

/** creo l'hash della chiave pubblica
  * PK dell'Information Object
    * da inserire nel campo A
    * dell'Identificatore */
byte[] improntaHashO = secureApi.hash
(secureApi.PK_O.getEncoded(),"SHA-1");

 /** assegno l'hash della chiave
   * pubblica al campo A di ID */
 String autenticatorO = DatatypeConverter.printBase64Binary
 (improntaHashO);

/** creo l'identificatore del proprietario
  * per la proprietàädi identificazione
* PS: solo per uso di test,
* l'identificatore viene creato
* nella fase di creazione
* dell'oggetto da parte dell'applicazione,
* tenendo conto delle problematiche
* e della struttura necessaria
* per l'utilizzo dellea libreria LEMS */
String identificatoreO = new String();
identificatoreO = "<identificator><type><typeO>"+typeO+
"</typeO>"+<typeHashA>"+typeHashAO+"</typeHashA>"+
      "<typeHashL/></type><autenticator>"+autenticatorO+
      "</autenticator><tagL/></identificator>";

/** assegno la firma ottenuta nel campo
  * sign_sCertData dei metadati associativi
  * ricreado il metadato per IO dinamico */
metadata= "<?xml version="+apice+"1.0"+apice+
" encoding="+apice+"UTF-8"+apice+" standalone="+apice+
"no"+apice+"?><metadata><locator><ipv4>192.168.27.67</ipv4>"+
"<protocol>tcp</protocol><port>4000</port></locator>"+
      "<attribute><gps>43716164,10396492</gps>"+
      "<semantic>video</semantic><identificatorO>"
      "+identificatoreO+"</identificatorO></attribute>"+
      "<security><sign/><sign_O/><pk>"+keyPK+"</pk>"+
```

```
      "<pk_O>"+keyPKO+"</pk_O><typeHash>"+tipoHash+"
      "</typeHash><typeDSAlg>"+tipoDSAlg+"</typeDSAlg>"+
      "</security></metadata>";

  /** converto la stringa del metadato
    * in byte per poterla firmare */
  byte [] metaByte = DatatypeConverter.parseBase64Binary
  (metadata);

/** firmo il metadato facendo
  * l'hash SHA-1 e poi usando
  * l algoritmo DSA
    * per la firma digitale utilizzando
    * la chiave privata SK_IO,
    * la firma ottenuta viene inserita nel
    * campo sign_s-certData dei metadati associativi */
byte[] sign_metadata = secureApi.DSAlg
(secureApi.SK_, metaByte, tipoDSAlg);

/** converto in stringa la firma
  * per inserirla in formato xml nei metadati */
f_sCertData = DatatypeConverter.printBase64Binary
(sign_metadata);

/** firmo il metadato facendo
  * l'hash SHA-1 e poi usando
  * l'algoritmo DSA
    * per la firma digitale utilizzando
    * la chiave privata SK_O,
    * la firma ottenuta viene inserita
    * nel campo sign_s-certData dei metadati associativi */
byte[] sign_metadataO = secureApi.DSAlg
(secureApi.SK_O, metaByte, tipoDSAlg);

/** converto in stringa la firma
  * per inserirla in formato
  * xml nei metadati */
f_sCertDataO = DatatypeConverter.printBase64Binary
(sign_metadataO);

/** assegno la firma ottenuta nel
  * campo sign_sCertData dei metadati associativi
  * ricreado il metadato per IO dinamico */
metadata= "<?xml version="+apice+"1.0"+apice+
" encoding="+apice+"UTF-8"+apice+"+ standalone="+
apice+"no"+apice+"?><metadata>"+
"<locator><ipv4>192.168.27.67</ipv4>"+
"<protocol>tcp</protocol><port>4000</port>"+
"</locator><attribute><gps>43716164,10396492</gps>"+
```

```
      "<semantic>video</semantic><identificatorO>"+
      +identificatoreO+"</identificatorO></attribute>"+
      "<security><sign>"+f_sCertData+"</sign><sign_O>"+
      +f_sCertDataO+"</sign_O><pk>"+keyPK+"</pk>"+
      "<pk_O>"+keyPKO+"</pk_O><typeHash>"+tipoHash+"</typeHash>"+
      "<typeDSAlg>"+tipoDSAlg+"</typeDSAlg></security></metadata>";
}

/** creo l'identificatore da pubblicare
 * PS: solo per uso di test,
 * l'identificatore viene creato nella
 * fase di creazione
 * dell'oggetto da parte dell'applicazione,
 * tenendo conto delle problematiche e della struttura
 * necessaria per l'utilizzo dellea libreria LEMS */
identifier.IdentifierS_ = "<?xml version="+apice+"1.0"+apice+
" encoding="+apice+"UTF-8"+apice+"+ standalone="+apice+
"no"+apice+"?><identificator>"+
"<type><typeIO>"+typeIO+"</typeIO><typeHashA>"
+typeHashA+"</typeHashA><typeHashL>"+typeHashL+"</typeHashL>"+
"</type><autenticator>"+autenticator+"</autenticator>"+
"<tagL>"+tagL+"</tagL></identificator>";

secureApi.publish(identifier.IdentifierS_,metadata);

Vector<Metadata> res = new Vector<Metadata>();
secureApi.get(identifier.IdentifierS_, res);

for(int i = 0; i < res.size(); ++i)
{
System.out.println(res.elementAt(i).getMetadata());
}

secureApi.unpublish(identifier.IdentifierS_, metadata);
}
}
```